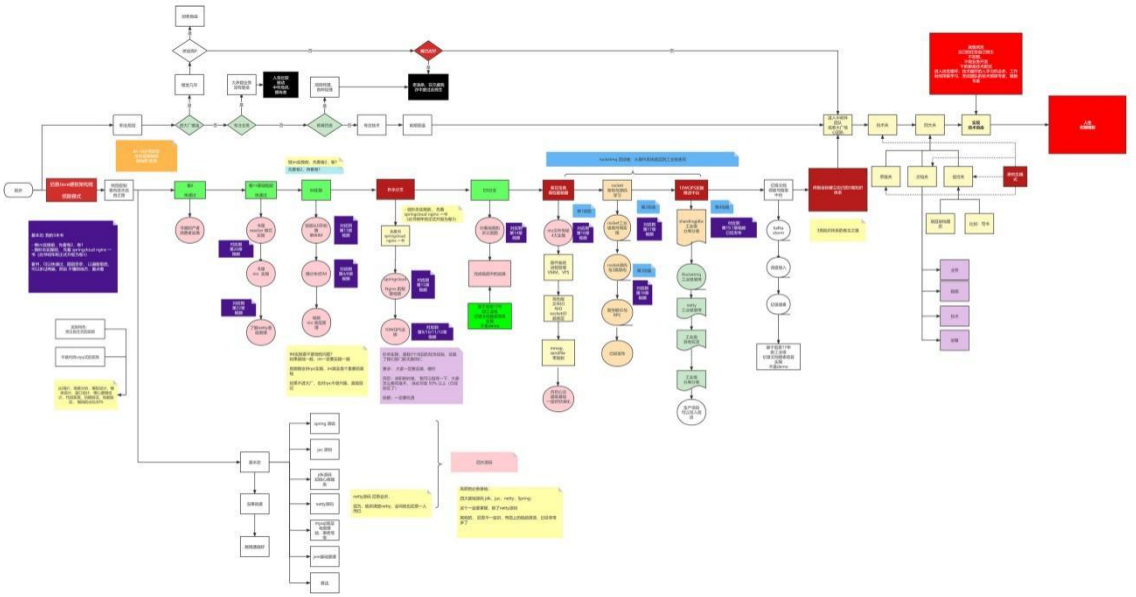
牛逼的职业发展之路

40 岁老架构尼恩用一张图揭秘：Java 工程师的高端职业发展路径，走向食物链顶端的之路

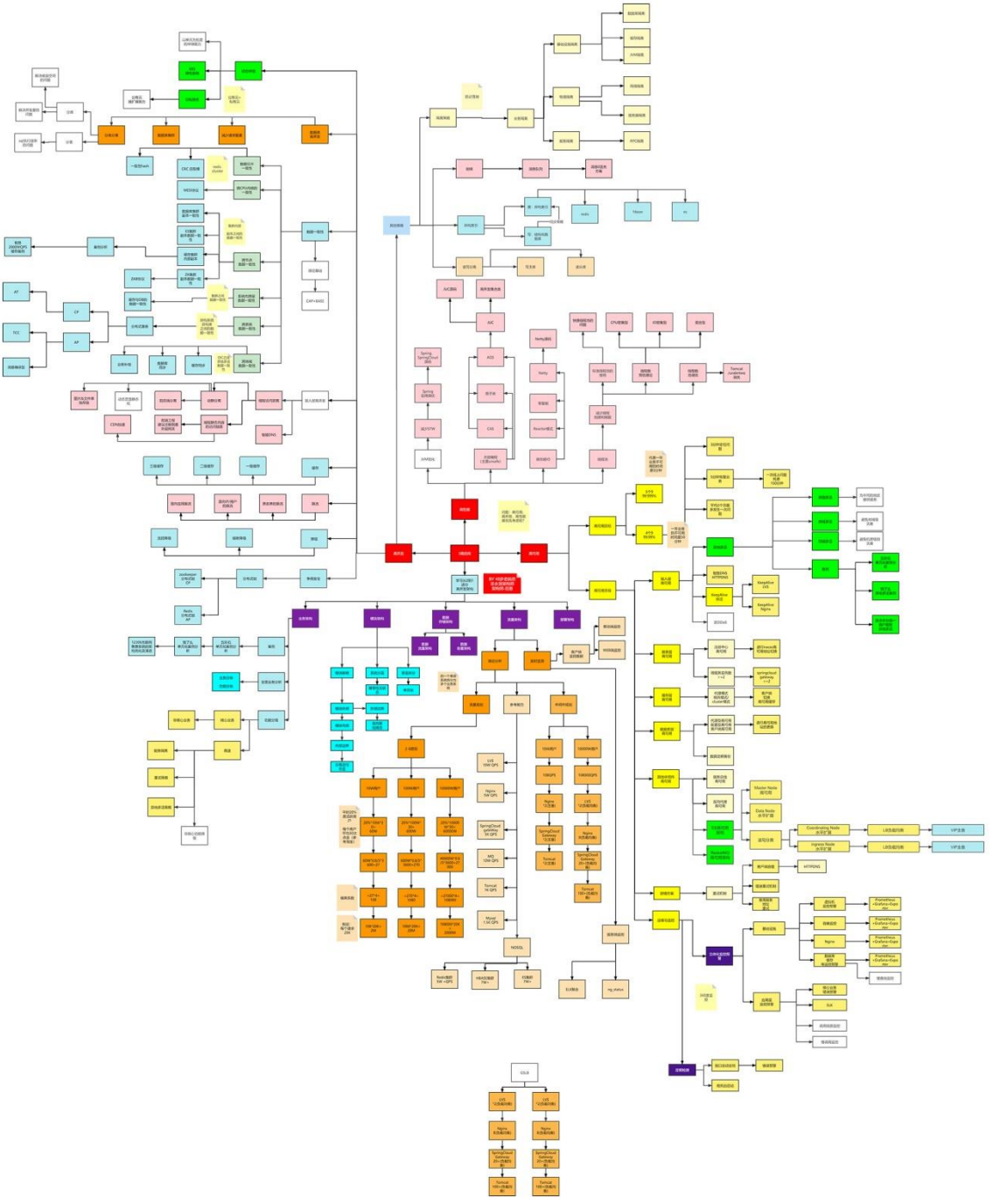
链接：https://www.processon.com/view/link/618a2b62e0b34d73f7eb3cd7



史上最全：价值10W的架构师知识图谱

此图梳理于尼恩的多个 3 高生产项目：多个亿级人民币的大型 SAAS 平台和智慧城市项目

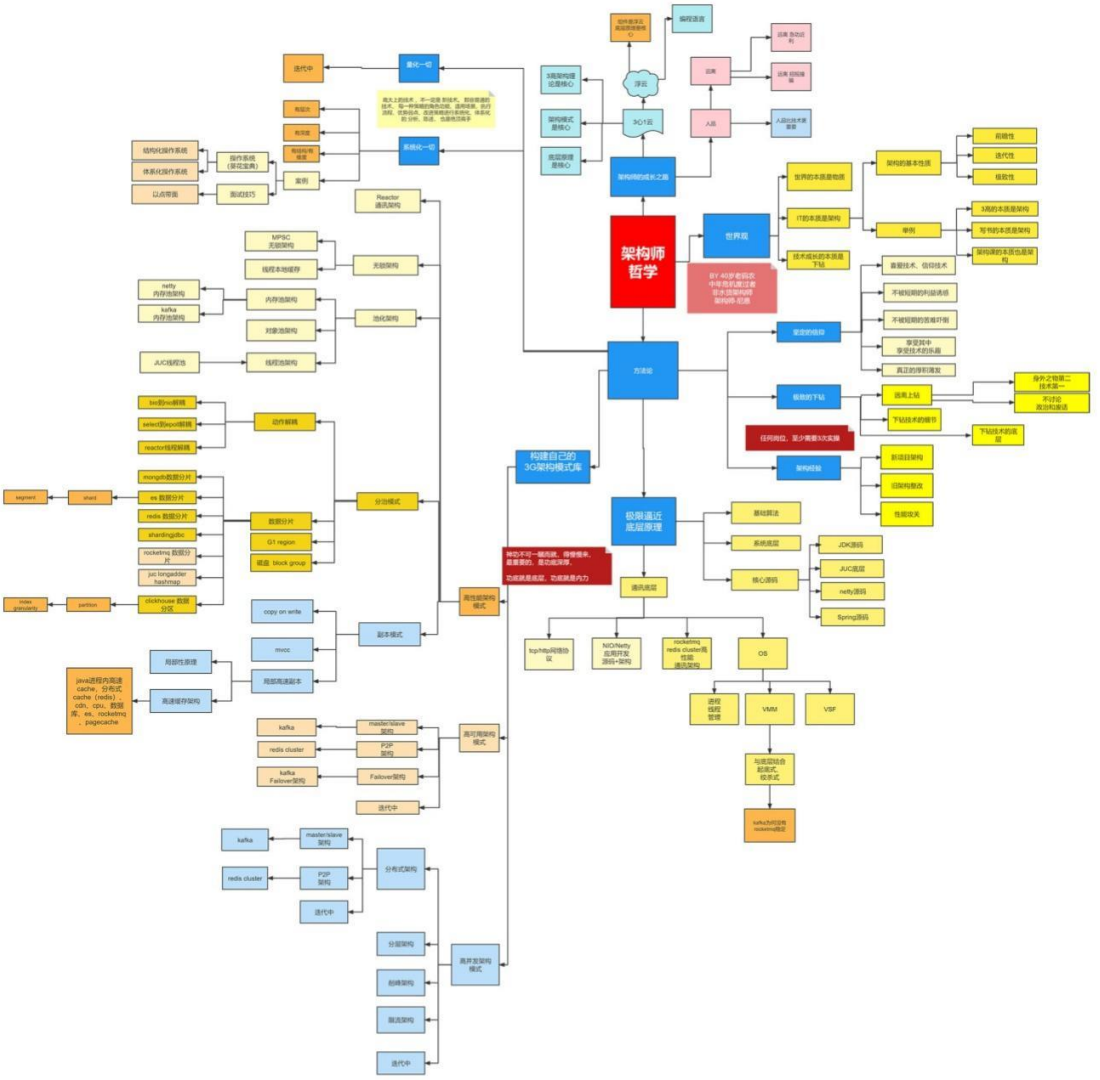
链接：https://www.processon.com/view/link/60fb94216376897l9d246739



牛逼的架构师哲学

40 岁老架构师尼恩对自己的 20 年的开发、架构经验总结

链接：https://www.processon.com/view/link/616f801963768961e9d9aec8



牛逼的3高架构知识宇宙

尼恩 3 高架构知识宇宙，帮助大家穿透 3 高架构，走向技术自由，远离中年危机

链接：https://www.processon.com/view/link/635097d2e0b34d40be778ab4

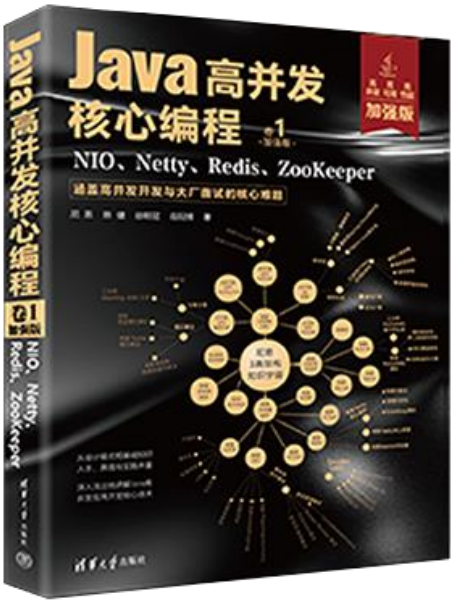
尼恩Java高并发三部曲（卷1加强版）

老版本：《Java 高并发核心编程 卷 1：NIO、Netty、Redis、ZooKeeper》（已经过时，不建议购买）

新版本：《Java 高并发核心编程 卷 1 **加强版**：NIO、Netty、Redis、ZooKeeper》

- 由浅入深地剖析了高并发 IO 的底层原理。
- 图文并茂的介绍了 TCP、HTTP、WebSocket 协议的核心原理。
- 细致深入地揭秘了 Reactor 高性能模式。
- 全面介绍了 Netty 框架，并完成单体 IM、分布式 IM 的实战设计。
- 详尽地介绍了 ZooKeeper、Redis 的使用，以帮助提升高并发、可扩展能力

详情：https://www.cnblogs.com/crazymakercircle/p/16868827.html



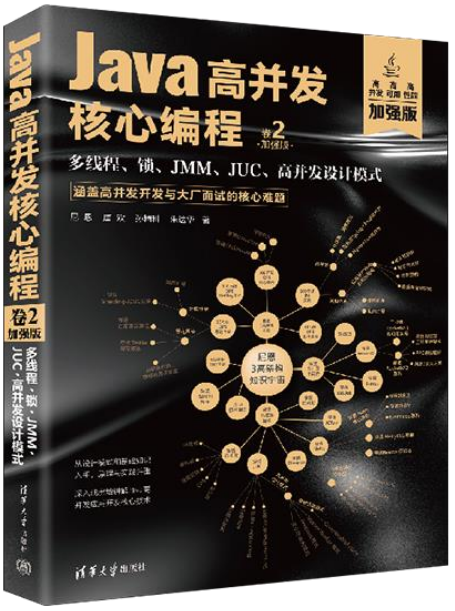
尼恩Java高并发三部曲（卷2加强版）

老版本：《Java 高并发核心编程 卷 2：多线程、锁、JMM、JUC、高并发设计模式》（已经过时，不建议购买）

新版本：《Java 高并发核心编程 卷 2 **加强版**：多线程、锁、JMM、JUC、高并发设计模式》

- 由浅入深地剖析了 Java 多线程、线程池的底层原理。
- 总结了 IO 密集型、CPU 密集型线程池的线程数预估算法。
- 图文并茂的介绍了 Java 内置锁、JUC 显式锁的核心原理。
- 细致深入地揭秘了 JMM 内存模型。
- 全面介绍了 JUC 框架的设计模式与核心原理，并完成其高核心组件的实战介绍。
- 详尽地介绍了高并发设计模式的使用，以帮助提升高并发、可扩展能力

详情参阅：https://www.cnblogs.com/crazymakercircle/p/16868827.html



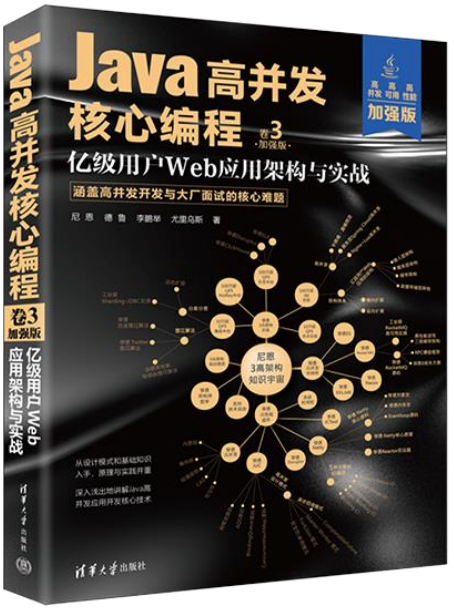
尼恩Java高并发三部曲（卷3加强版）

老版本：《SpringCloud Nginx 高并发核心编程》（已经过时，不建议购买）

新版本：《Java 高并发核心编程 卷 3 **加强版**：亿级用户 Web 应用架构与实战》

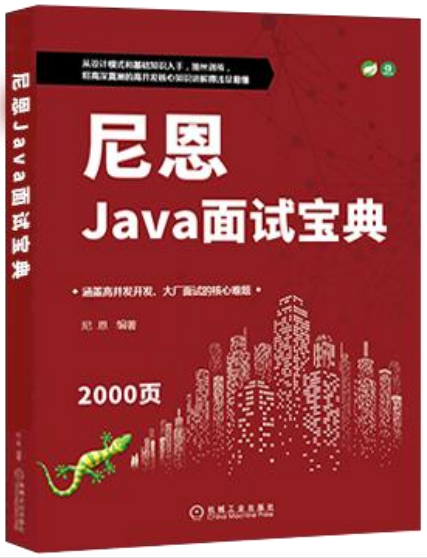
- 在当今的面试场景中，3 高知识是大家面试必备的核心知识，本书基于亿级用户 3 高 Web 应用的架构分析理论，为大家对 3 高架构系统做一个系统化和清晰化的介绍。
- 从 Java 静态代理、动态代理模式入手，抽丝剥茧地解读了 Spring Cloud 全家桶中 RPC 核心原理和执行过程，这是高级 Java 工程师面试必备的基础知识。
- 从Reactor 反应器模式入手，抽丝剥茧地解读了Nginx 核心思想和各配置项的底层知识和原理，这是高级 Java 工程师、架构师面试必备的基础知识。
- 从观察者模式入手，抽丝剥茧地解读了 RxJava、Hystrix 的核心思想和使用方法，这也是高级 Java 工程师、架构师面试必备的基础知识。

详情：https://www.cnblogs.com/crazymakercircle/p/16868827.html



尼恩Java面试宝典

35 个专题（卷王专供+ 史上最全 + 2023 面试必备）
详情：https://www.cnblogs.com/crazymakercircle/p/13917138.html



网盘 › 面试 › 尼恩Java面试宝典

名称

专题01：JVM面试题（卷王专供 + 史上最全 + 2023面试必备）-V10-from-尼恩Java面试宝典-release.pdf
专题02：Java算法面试题（卷王专供 + 史上最全 + 2023面试必备）-V2 -from-Java面试红宝书-release.pdf
专题03：Java基础面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书 -release.pdf
专题04：架构设计面试题（卷王专供+ 史上最全 + 2023面试必备）-V10-from-Java面试红宝书-release.pdf
专题05：Spring面试题_专题06：SpringMVC_专题07：Tomcat面试题（卷王专供+ 史上最全 + 2023面试必备）-V3-from-尼恩面试宝典-release.pdf
专题08：SpringBoot面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题09：网络协议面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题10：TCPIP协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题11：JUC并发包与容器类（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题12：设计模式面试题 （卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题13：死锁面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题14：Redis 面试题 （卷王专供+ 史上最全 + 2023面试必备）-V5-from-Java面试红宝书-release.pdf
专题15：分布式锁 面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题16：Zookeeper 面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题17：分布式事务面试题（卷王专供 + 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题18：一致性协议 （卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题19：Zab协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题20：Paxos 协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题21：raft 协议（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题22：Linux面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题23：Mysql 面试题（卷王专供+ 史上最全 + 2023面试必备）-V11-from-Java面试红宝书-release.pdf
专题24：SpringCloud 面试题（卷王专供+ 史上最全 + 2023面试必备）-V12-from-Java面试红宝书-release.pdf
专题25：Netty 面试题 （卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题26：消息队列面试题：RabbitMQ、Kafka、RocketMQ（卷王专供+ 史上最全 + 2023面试必备）-V10-from-Java面试红宝书-release.pdf
专题27：内存泄漏 内存溢出（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题28：JVM 内存溢出 实战（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题29：多线程面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题30：HR面试题：过五关斩六将后，小心阴沟翻船！（史上最全、避坑宝典）-V2-from-Java面试红宝书-release.pdf
专题31：Hash连环炮面试题（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题32：大厂面试的基本流程和面试准备（阿里、腾讯、网易、京东、头条.....）-V2-from-Java面试红宝书-release.pdf
专题33：BST、AVL、RBT红黑树、三大核心数据结构（卷王专供+ 史上最全 + 2023面试必备）-V2-from-Java面试红宝书-release.pdf
专题34：Elasticsearch面试题（卷王专供+ 史上最全 + 2023面试必备）-V3-from-Java面试红宝书-release.pdf
专题35：Mybatis面试题（卷王专供+ 史上最全 + 2023面试必备）-V3-from-尼恩Java面试宝典-release.pdf

专题28：JVM 内存溢出 实战（史上最全、定期更新）

本文版本说明：V2

> 此文的格式，由markdown 通过程序转成而来，由于很多表格，没有来的及调整，出现一个格式问题，尼恩在此给大家道歉啦。
>
> 由于社群很多小伙伴，在面试，不断的交流最新的面试难题，所以，《Java面试红宝书》， 后面会不断升级，迭代。
>
> 本专题，作为 《Java面试红宝书》专题之一， 《Java面试红宝书》一共**30个面试专题，后续还会增加**

###

V16版本升级说明：

聊聊：如何进行内存溢出OOM实操分析

聊聊：如何进行内存泄漏实操分析

《Java面试红宝书》升级的规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以扫描扫架构师尼恩微信， 发送 “领取电子书” 获取。

尼恩的微信二维码在哪里呢？ 请参见文末

面试问题交流说明：

如果遇到面试难题，或者职业发展问题， 或者中年危机问题，都可以来 疯狂创客圈社群交流，

加入交流群，加尼恩微信即可，

入交流群，加尼恩微信即可， 发送**“入群”**

JVM性能优化面试题

JVM内存区域常见问题

Java 中会存在内存泄漏吗，简述一下?

Java 内存分配?

Java 堆的结构是什么样子的?

什么是堆中的永久代（Perm Gen space）?

简述各个版本内存区域的变化?

说说各个区域的作用?

JVM的执行子系统常见问题

Java 类加载过程?

描述一下 JVM 加载 Class 文件的原理机制?什么是类加载器?

类加载器有哪些?

类加载器双亲委派模型机制?

垃圾回收常见问题

什么是GC?

为什么要有 GC?

简述一下Java 垃圾回收机制?

如何判断一个对象是否存活?

垃圾回收的优点和原理，并考虑 2 种回收机制?

垃圾回收器的基本原理是什么?

垃圾回收器可以马上回收内存吗?

有什么办法主动通知虚拟机进行垃圾回收?

深拷贝和浅拷贝?

System.gc() 和 Runtime.gc() 会做些什么?

如果对象的引用被置为 null，垃圾收集器是否会立即释放对象占用的内存?

什么是分布式垃圾回收（DGC）?

它是如何工作的?

串行（serial）收集器和吞吐量（throughput）收集器的区别是什么?

在 Java 中，对象什么时候可以被垃圾回收？简述Minor GC 和 Major GC？JVM 的永久代中会发生垃圾回收么？Java 中垃圾收集的方法有哪些?

性能优化常见问题

讲讲你理解的性能评价及测试指标?

常用的性能优化方式有哪些?

什么是GC调优?

JVM调优工具

Jconsole，jProfile，VisualVM

Jconsole : jdk自带，功能简单，但是可以在系统有一定负荷的情况下使用。对垃圾回收算法有很详细的跟踪。详细说明参考这里

JProfiler：商业软件，需要付费。功能强大。详细说明参考这里

VisualVM：JDK自带，功能强大，与JProfiler类似。推荐。

本文章向大家介绍JVM调优工具详解，主要内容包括前言、jdk自带工具、一、Jmap、1.2 jmap -histo、1.3 jmap -heap、1.4 jmap -dump、二、Jstack、三、Jinfo、四、Jstat、垃圾回收统计、JVM运行情况预估、内存泄露到底是怎么回事、五、Arthas、dashboard、thread、总结、基本概念、基础应用、原理机制和需要注意的事项等，并结合实例形式分析了其使用技巧，希望通过本文能帮助到大家理解应用这部分内容。

jps

事先启动一个web应用程序，用jps查看其进程id，接着用各种jdk自带命令优化应用

jmap -histo

可以使用"jmp -histo **进程id**" 此命令可以用来查看**内存信息，实例个数以及占用内存大小**

打开log.txt，文件内容如下：

- num：序号
- instances：实例数量
- bytes：占用空间大小
- class name：类名称，[C is a char[]，[S is a short[]，[I is a int[]，[B is a byte[]，[[I is a int[][]

jmap -heap

查看对应**进程id**堆信息可以看到下面打印出堆的配置最大小容量等及Eden区From和Old区使用情况

jmap -dump

可以直接使用命令导出选择进程ID的dump文件

> jmap -dump:format=b,file=eureka.hprof 14660

1. -XX:+HeapDumpOnOutOfMemoryError
2. -XX:HeapDumpPath=./ （路径） 也可以设置内存溢出自动导出dump文件(内存很大的时候，可能会导不出来)

代码如下：

```
package com.jvm;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

public class OOMTest {
```

```
    public static List<Object> list = new ArrayList<>();

    // JVM设置
    // -Xms10M -Xmx10M -XX:+PrintGCDetails -XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=D:jvm.dump
    public static void main(String[] args) {

        List<Object> list = new ArrayList<>();
        int i = 0;
        int j = 0;

        while (true) {

            list.add(new User(i++, UUID.randomUUID().toString()));
            new User(j--, UUID.randomUUID().toString());
        }
    }
}
```

当然也可以用jvisualvm命令工具导入该dump文件分析使用jvisualvm就更简单了只需要在cmd下输入jvisualvm就可以打开

导入输出的dump文件

Jstack

用jstack加进程id查找死锁，代码见如下示例：

```
package com.jvm;

public class DeadLockTest {


    private static Object lock1 = new Object();
    private static Object lock2 = new Object();

    public static void main(String[] args) {

        new Thread(() ->{

            synchronized (lock1) {

                try {

                    System.out.println("thread1 begin");
                    Thread.sleep(5000);
                } catch (InterruptedException e) {


                }
                synchronized (lock2) {

                    System.out.println("thread1 end");
                }
```

```
            }
        }).start();

        new Thread(() ->{

            synchronized (lock2) {

                try {

                    System.out.println("thread2 begin");
                    Thread.sleep(5000);
                } catch (InterruptedException e) {


                }
                synchronized (lock1) {

                    System.out.println("thread2 end");
                }
            }
        }).start();

        System.out.println("main thread end");
    }


}
```

运行后打印线程详细信息

Thread-1" 线程名 prio=5 优先级=5 tid=0x000000001fa9e000 线程id nid=0x2d64 线程对应的本地线程标识nid java.lang.Thread.State: BLOCKED 线程状态

图上详细的说明了Thread-0与Thread-1相互引用而造成死锁就连代码出现的行号也是打印出来了。

还可以用jvisualvm自动检测死锁只需要查看对应运行java的进程即可

Jinfo

查看正在运行的Java应用程序的扩展参数 查看jvm的参数

查看jvm的参数 jinfo -flags **进程ID**

查看java系统参数 jinfo -sysprops**进程ID**

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。

命令的格式如下： jstat [-命令选项] [vmid] [间隔时间(毫秒)] [查询次数] 注意：使用的jdk版本是jdk8

jstat -gc垃圾回收统计

jstat -gc pid 最常用，可以评估程序内存使用及GC压力整体情况

- S0C：第一个幸存区的大小，单位KB
- S1C：第二个幸存区的大小
- S0U：第一个幸存区的使用大小
- S1U：第二个幸存区的使用大小
- EC：伊甸园区的大小
- EU：伊甸园区的使用大小
- OC：老年代大小
- OU：老年代使用大小
- MC：方法区大小(元空间)
- MU：方法区使用大小
- CCSC:压缩类空间大小
- CCSU:压缩类空间使用大小
- YGC：年轻代垃圾回收次数
- YGCT：年轻代垃圾回收消耗时间，单位s
- FGC：老年代垃圾回收次数
- FGCT：老年代垃圾回收消耗时间，单位s
- GCT：垃圾回收消耗总时间，单位s

JVM运行情况预估

用 jstat gc -pid 命令可以计算出如下一些关键数据，有了这些数据就可以采用之前介绍过的优化思路，先给自己的系统设置一些初始性的 JVM参数，比如堆内存大小，年轻代大小，Eden和Survivor的比例，老年代的大小，大对象的阈值，大龄对象进入老年代的阈值等。

年轻代对象增长的速率

可以执行命令 jstat -gc pid 1000 10 (每隔1秒执行1次命令，共执行10次)，通过观察EU(eden区的使用)来估算每秒eden大概新增多少对 象，如果系统负载不高，可以把频率1秒换成1分钟，甚至10分钟来观察整体情况。注意，一般系统可能有高峰期和日常期，所以需要在不 同的时间分别估算不同情况下对象增长速率。

Young GC的触发频率和每次耗时

知道年轻代对象增长速率我们就能推根据eden区的大小推算出Young GC大概多久触发一次，Young GC的平均耗时可以通过 YGCT/YGC 公式算出，根据结果我们大概就能知道**系统大概多久会因为Young GC的执行而卡顿多久。**

每次Young GC后有多少对象存活和进入老年代

这个因为之前已经大概知道Young GC的频率，假设是每5分钟一次，那么可以执行命令 jstat -gc pid 300000 10 ，观察每次结果eden， survivor和老年代使用的变化情况，在每次gc后eden区使用一般会大幅减少，survivor和老年代都有可能增长，这些增长的对象就是每次 Young GC后存活的对象，同时还可以看出每次Young GC后进去老年代大概多少对象，从而可以推算出**老年代对象增长速率。**

Full GC的触发频率和每次耗时

知道了老年代对象的增长速率就可以推算出Full GC的触发频率了，Full GC的每次耗时可以用公式 FGCT/FGC 计算得出。

优化思路其实简单来说就是尽量让每次Young GC后的存活对象小于Survivor区域的50%，都留存在年轻代里。尽量别让对象进入老年 代。尽量减少Full GC的频率，避免频繁Full GC对JVM性能的影响。

内存泄露到底是怎么回事

再给大家讲一种情况，一般电商架构可能会使用多级缓存架构，就是redis加上JVM级缓存，大多数同学可能为了图方便对于JVM级缓存就 简单使用一个hashmap，于是不断往里面放缓存数据，

但是很少考虑这个map的容量问题，结果这个缓存map越来越大，一直占用着老 年代的很多空间，时间长了就会导致full gc非常频繁，这就是一种内存泄漏，

对于一些老旧数据没有及时清理导致一直占用着宝贵的内存 资源，

时间长了除了导致full gc，还有可能导致OOM。

这种情况完全可以考虑采用一些成熟的JVM级缓存框架来解决，比如Caffiene 等自带一些LRU数据淘汰算法的框架来作为JVM级的缓存。

Arthas

Arthas 是 Alibaba 在 2018 年 9 月开源的 Java 诊断工具。

支持 JDK6+， 采用命令行交互模式，可以方便的定位和诊断 线上程序运行问题。

Arthas 官方文档十分详细，详见：https://arthas.aliyun.com/en-us/

Arthas使用

> github下载arthas wget https://alibaba.github.io/arthas/arthas-boot.jar
>
> 或者 Gitee 下载 wget https://arthas.gitee.io/arthas-boot.jar

用java -jar运行即可，可以识别机器上所有Java进程(我们这里之前已经运行了一个Arthas测试程序，代码见下方

```
package com.jvm;

import java.util.HashSet;

public class Arthas {


    private static HashSet hashSet = new HashSet();

    public static void main(String[] args) {

        // 模拟 CPU 过高
        cpuHigh();
        // 模拟线程死锁
        deadThread();
        // 不断的向 hashSet 集合增加数据
        addHashSetThread();
    }

    /** * 不断的向 hashSet 集合添加数据 */
    public static void addHashSetThread() {

        // 初始化常量
        new Thread(()-> {
```

```
            int count = 0;
            while (true){

                try {

                    hashSet.add("count" + count);
                    Thread.sleep(1000);
                    count++;
                }catch (InterruptedException e) {

                    e.printStackTrace();
                }
            }
        }).start();
    }

    public static void cpuHigh() {

        new Thread(()->{

            while (true){


            }
        }).start();
    }

    /** * 死锁 */
    private static void deadThread() {

        /** 创建资源 */
        Object resourceA = new Object();
        Object resourceB = new Object();

        Thread threadA = new Thread(()->{

           synchronized (resourceA){

               System.out.println(Thread.currentThread() + " get ResourceA");
               try {

                   Thread.sleep(1000);
               }catch (InterruptedException e) {

                   e.printStackTrace();
               }
               System.out.println(Thread.currentThread() + "waiting get
resourceB");
               synchronized (resourceB){

                   System.out.println(Thread.currentThread() + " get
resourceB");
               }
           }
        });
        Thread threadB = new Thread(()->{
```

```
            synchronized (resourceB){

                System.out.println(Thread.currentThread() + " get ResourceB");
                try {

                    Thread.sleep(1000);
                }catch (InterruptedException e) {

                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread() + "waiting get
resourceA");
                synchronized (resourceA){

                    System.out.println(Thread.currentThread() + " get
resourceA");
                }
            }
        });
        threadA.start();
        threadB.start();
    }
}
```

选择进程序号1，进入进程信息操作

dashboard

输入dashboard可以查看整个进程的运行情况，线程、内存、GC、运行环境信息：

thread

输入thread可以查看线程详细情况

输入 thread加上线程ID 可以查看线程堆栈

输入 thread -b 可以查看线程死锁

其实Arthas这个工具并没有大家想象的那么难

使用起来非常的简单

具体的命令可以用help命令查看，或查看文档：

https://arthas.aliyun.com/doc/advanced-use.html#id2

更多的还是需要自己去摸索

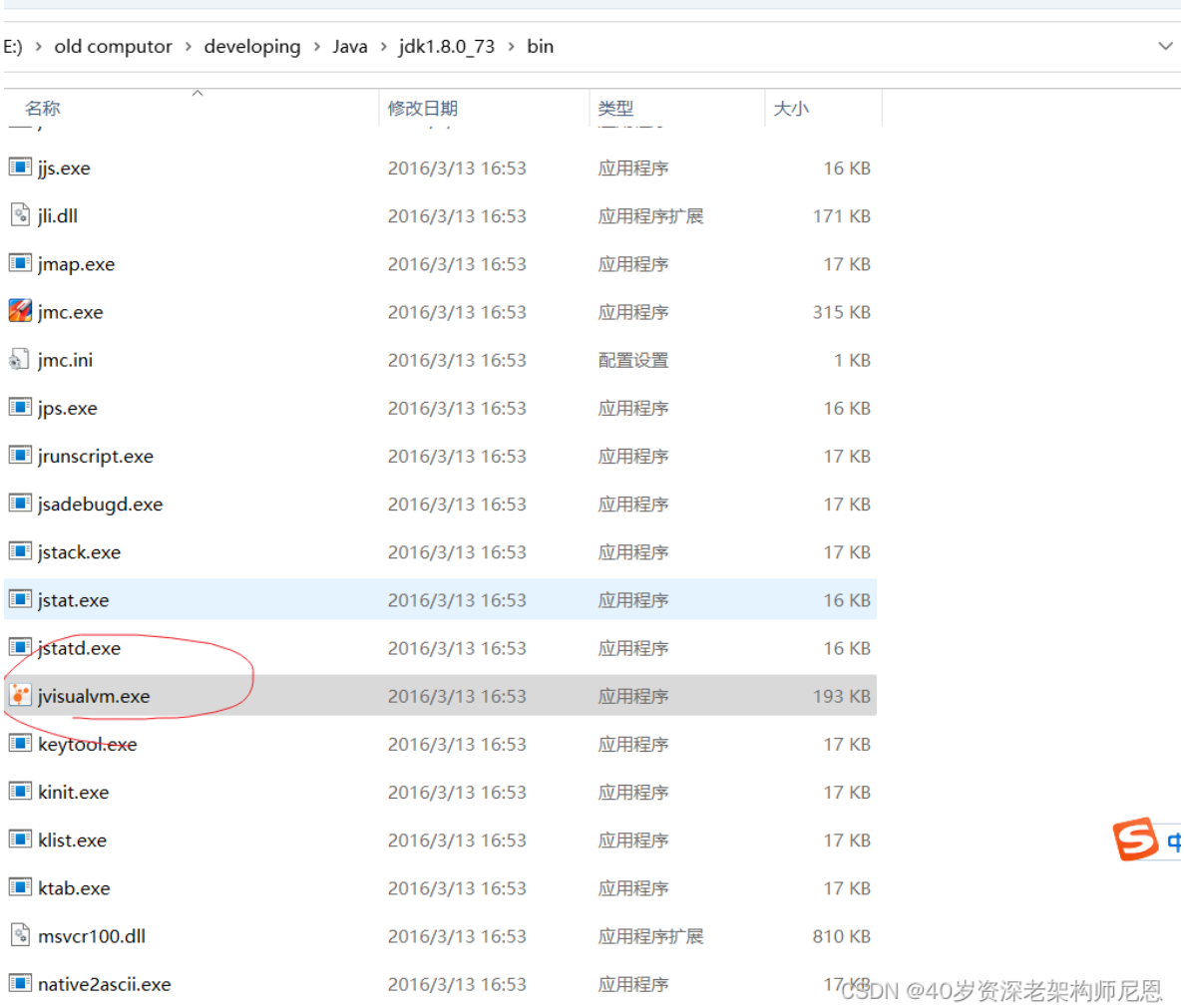
JVisualVM初探

VisualVM 是Netbeans的profile子项目，

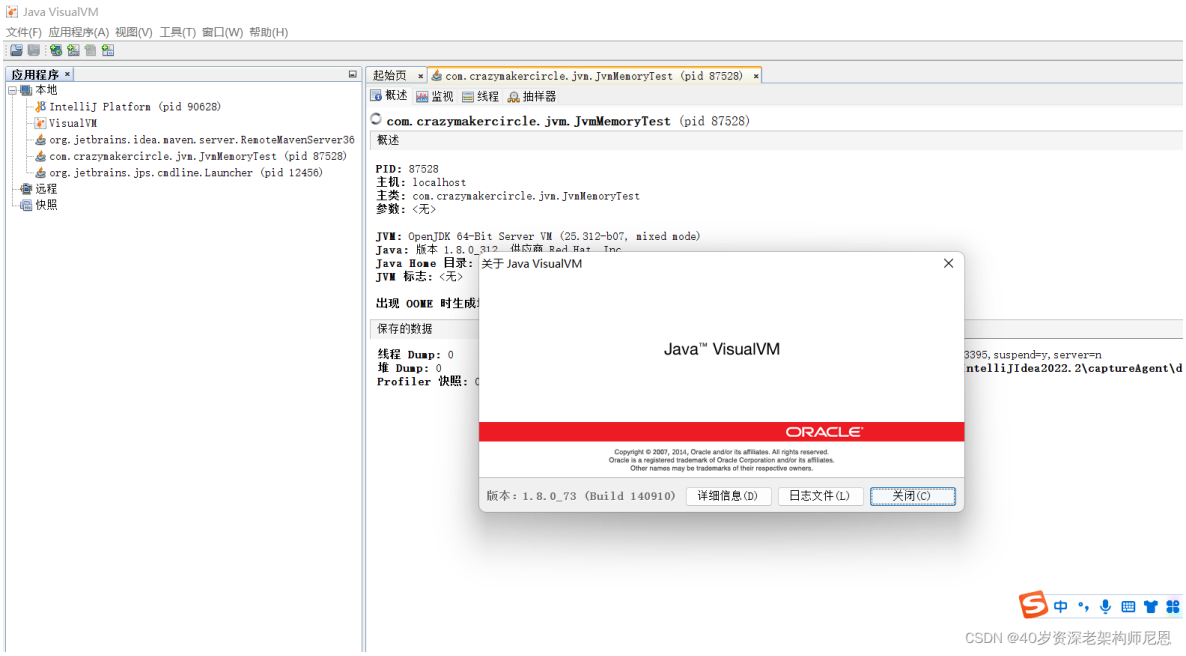
已在JDK6.0 update 7 中自带(java启动时不需要特定参数，监控工具在bin/jvisualvm.exe)，

能够监控线程，内存情况，查看方法的CPU时间和内存中的对 象，已被GC的对象，反向查看分配的堆栈(如100个String对象分别由哪几个对象分配出来的)。

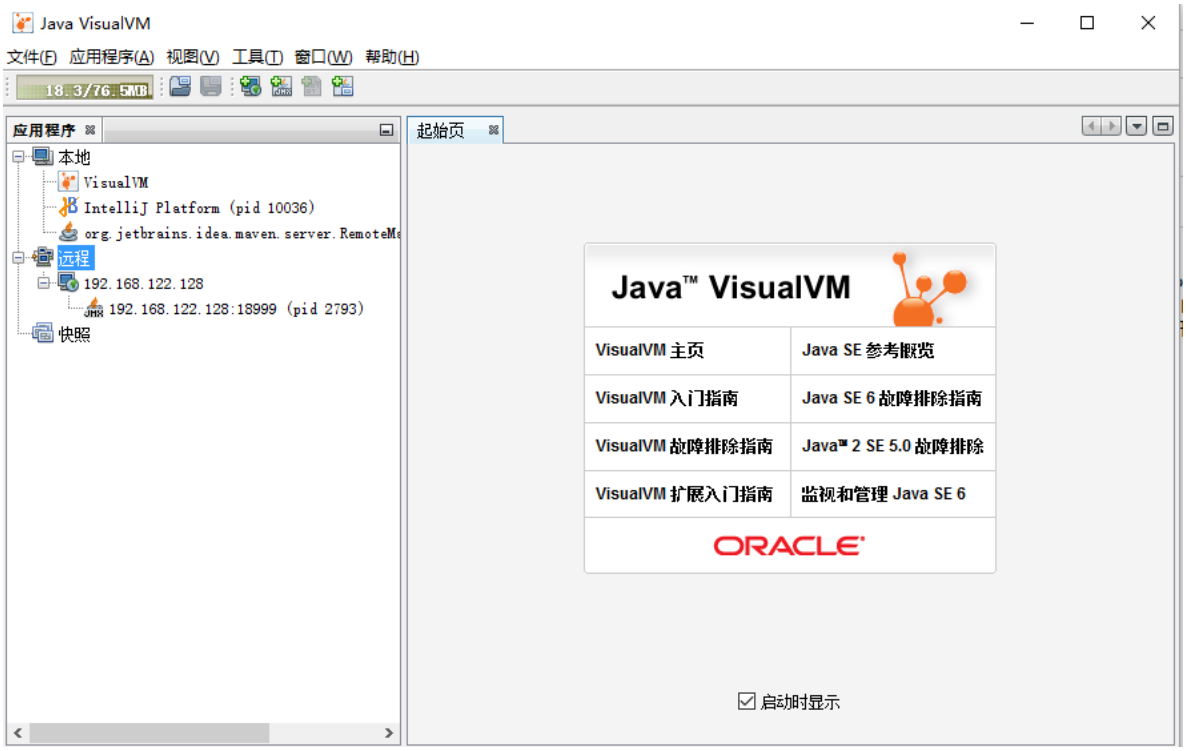
在JDK_HOME/bin 目录下面，有一个jvisualvm.exe文件，



双击打开，从UI上来看，这个软件是基于NetBeans开发的了。



其默认页面为：

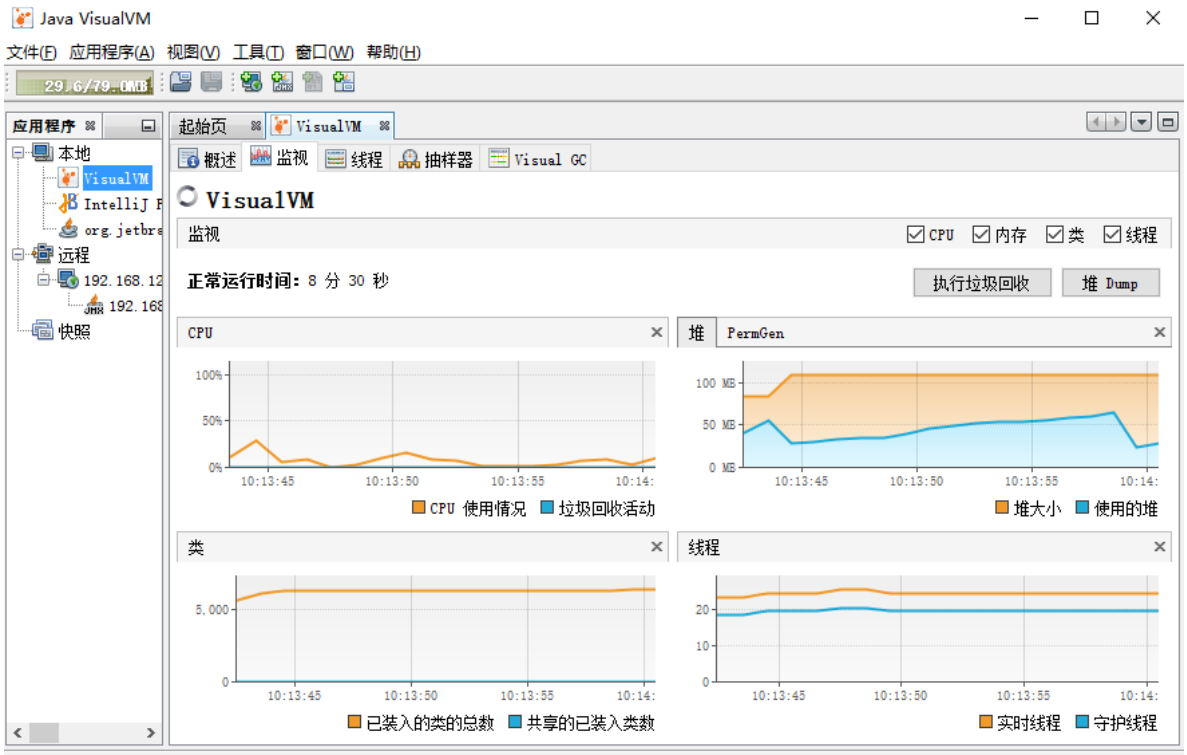


左侧分为本地和远程。

可以进行远程和本地监控

- 本地监控，打开VisualVM后，会直接检测到本地的Java 进程。
- 远程监控需要打开jmx，通过IP地址，进行远程的连接。

双击本地中VisualVM线程，可以看到如下监控内容：



具体的介绍参看:

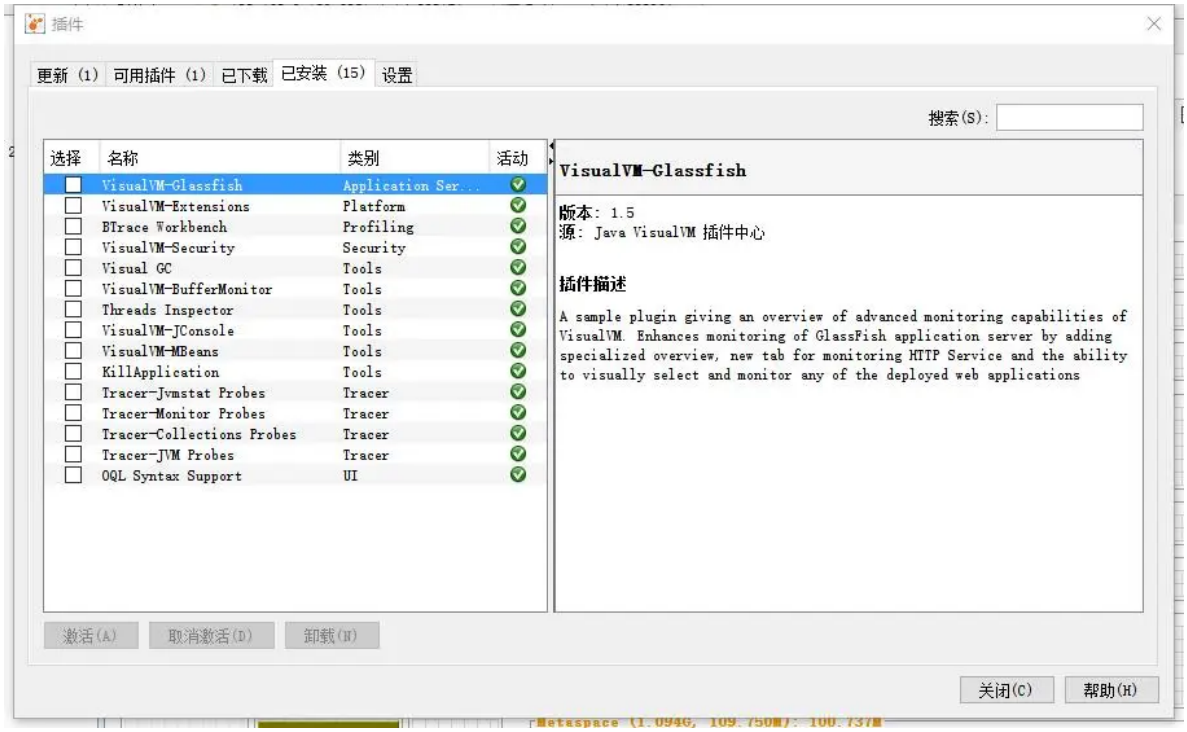
http://www.ibm.com/developerworks/cn/java/j-lo-visualvm/

VisualVM 插件

默认情况下，很多插件是没有的。 比如 virtual GC插件

VisualVM可以根据需要安装不同的插件，

每个插件的关注点都不同，有的主要监控GC，有的主要监控内存，有的监控线程等。



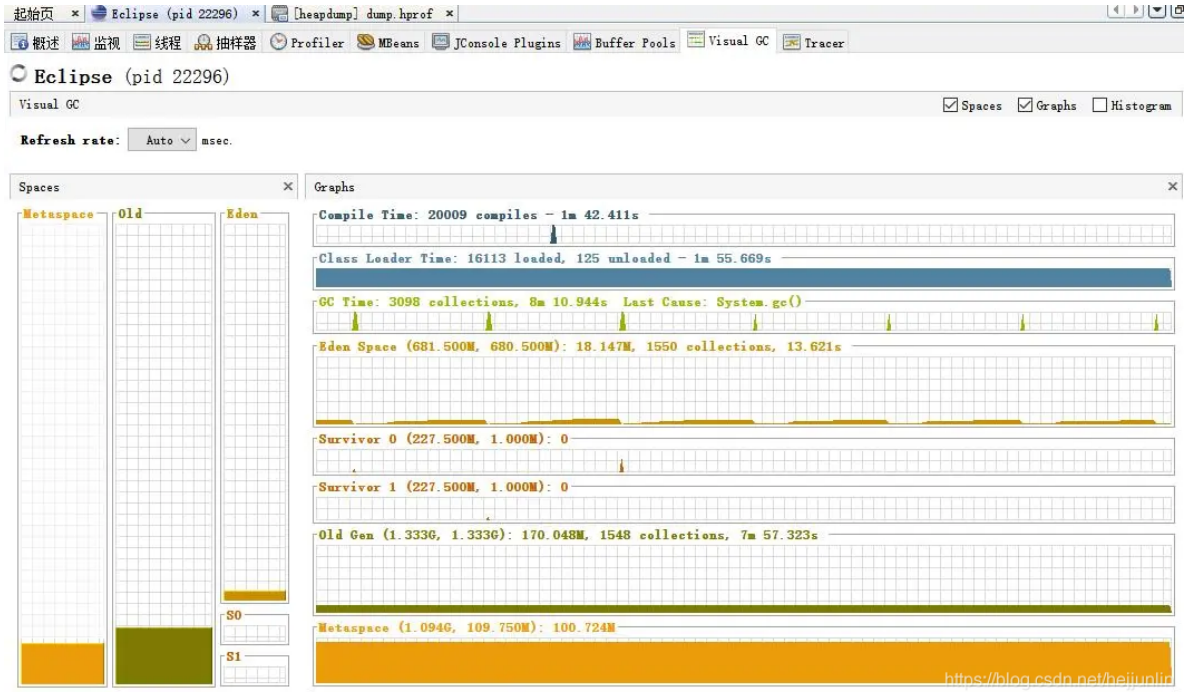
如何安装插件：

1、从主菜单中选择“工具”>“插件”。

2、在“可用插件”标签中，选中该插件的“安装”复选框。单击“安装”。

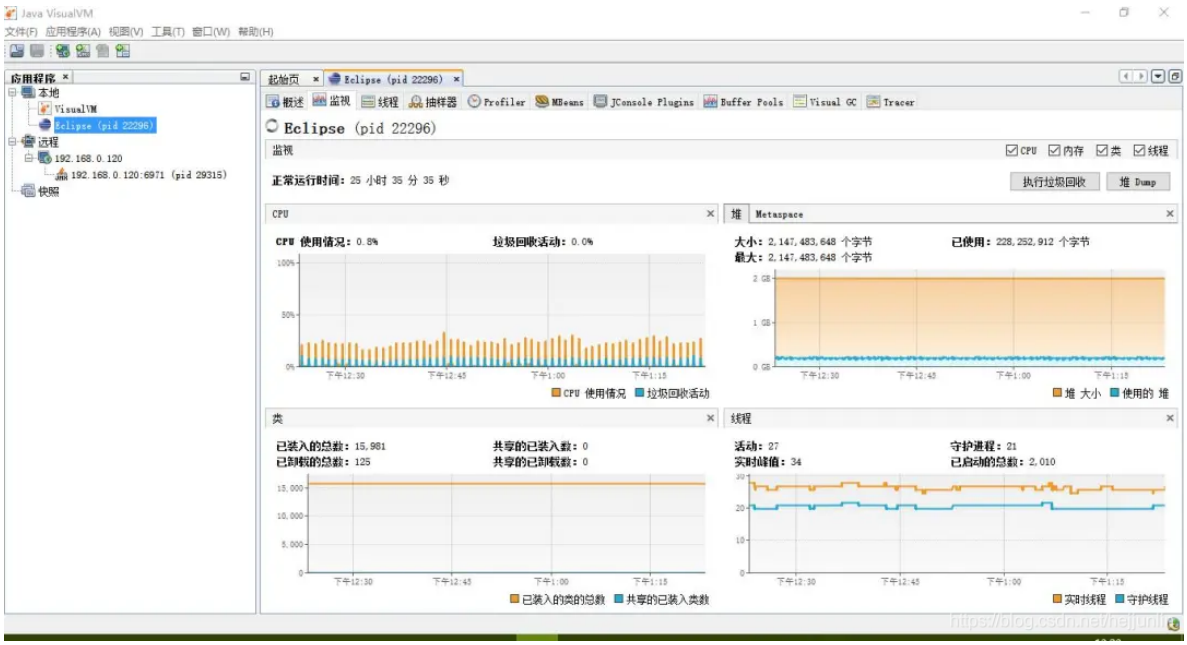
3、逐步完成插件安装程序。

我这里以 Eclipse(pid 22296)为例，双击后直接展开，主界面展示了系统和jvm两大块内容，点击右下方jvm参数和系统属性可以参考详细的参数信息.

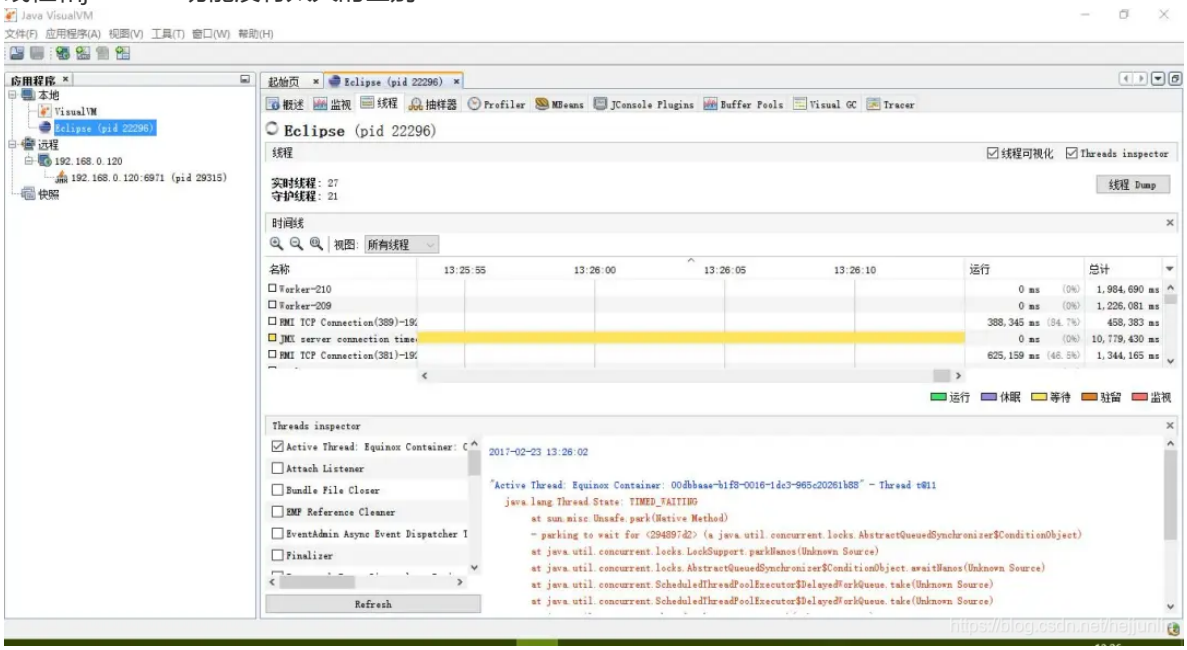


因为VisualVM的插件太多，我这里主要介绍三个我主要使用几个：监控、线程、Visual GC

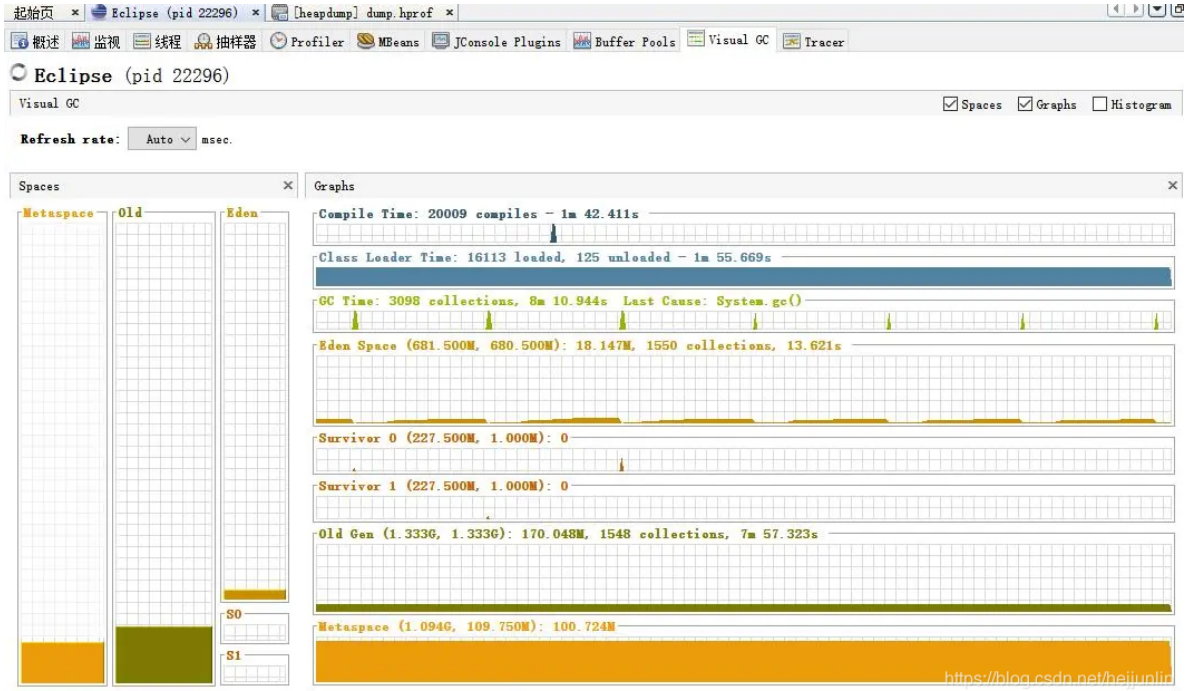
监控的主页其实也就是，cpu、内存、类、线程的图表



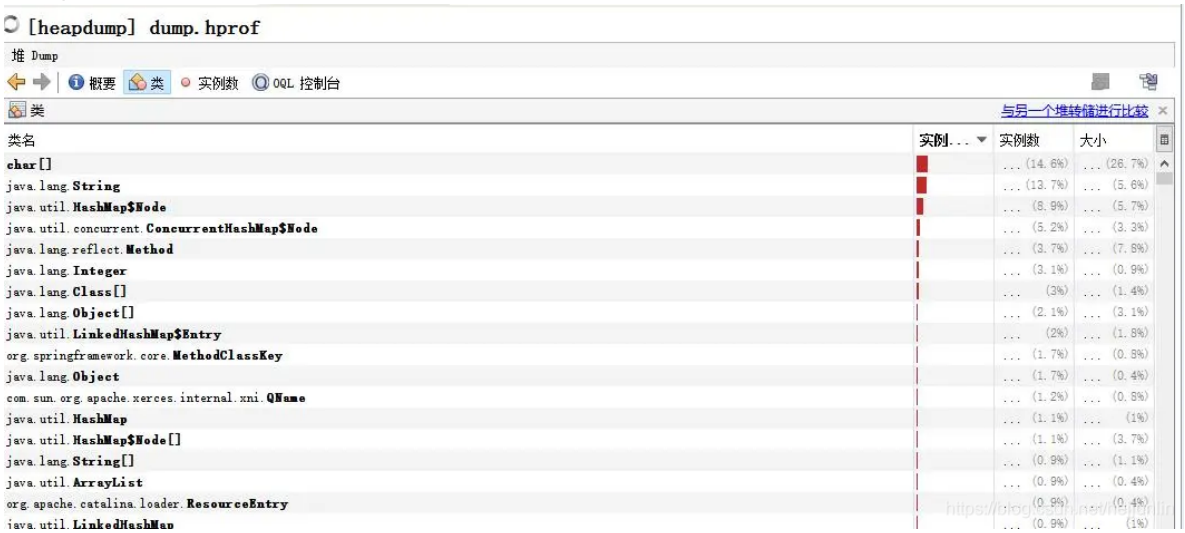
线程和jconsole功能没有太大的区别



Visual GC 是常常使用的一个功能，可以明显的看到年轻代、老年代的内存变化，以及gc频率、gc的时间等。

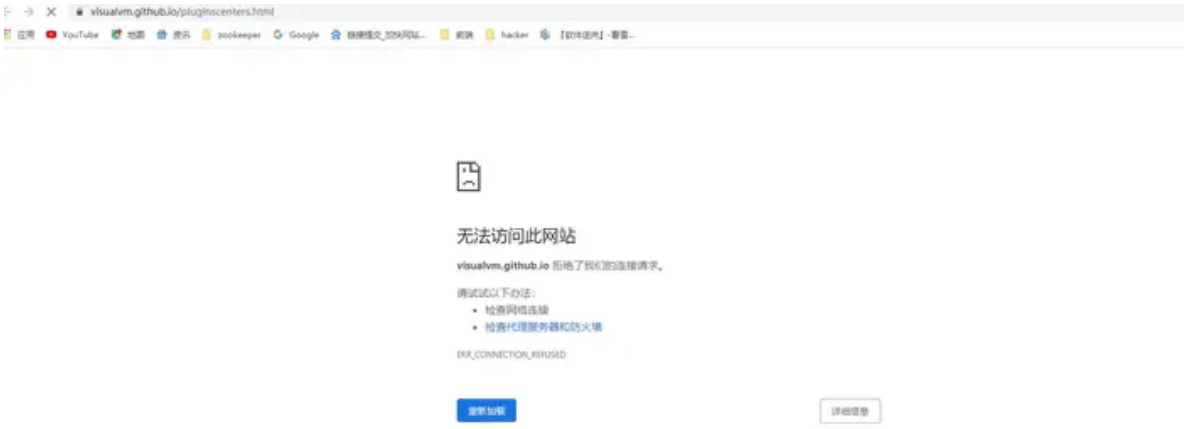


以上的功能其实jconsole几乎也有，VisualVM更全面更直观一些，另外VisualVM非常多的其它功能，可以分析dump的内存快照，
dump出来的线程快照并且进行分析等，还有其它很多的插件大家可以去探索



JDK自带的jvisualvm安装插件时报错

需要给jvisualvm添加个visual gc插件时，需要访问github.io但是发现访问如下图。



在线更新 插件 地址

无法连接Java VisualVM 插件中心,

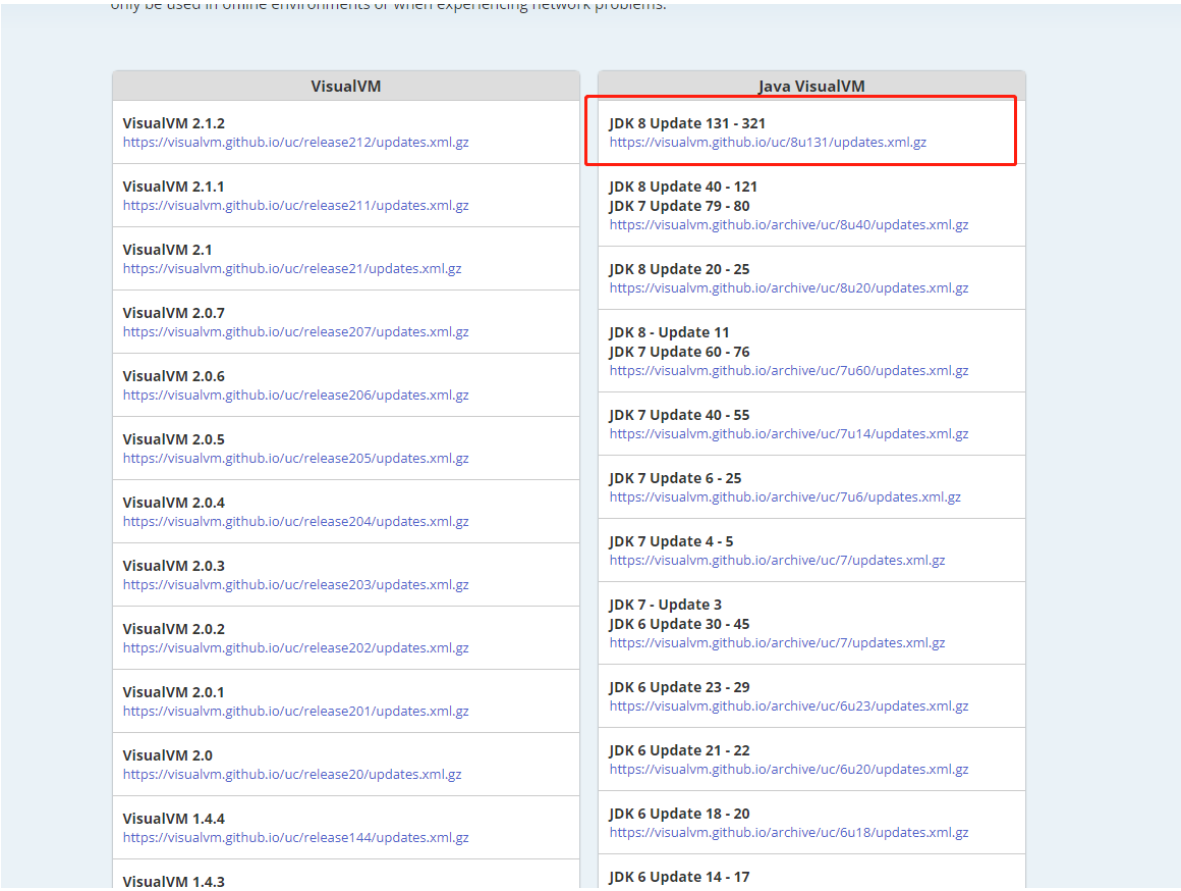
因为Zero sized file reported at

https://visualvm.java.net/uc/8u40/updates.xml.gz?unique=039f77b95-689f-47b0-96cd-62e51dd66456_be47e117-6183-4d0b-8579-66f262b40b35

先到https://visualvm.github.io/pluginscenters.html

找到与你jdk版本相对应的url

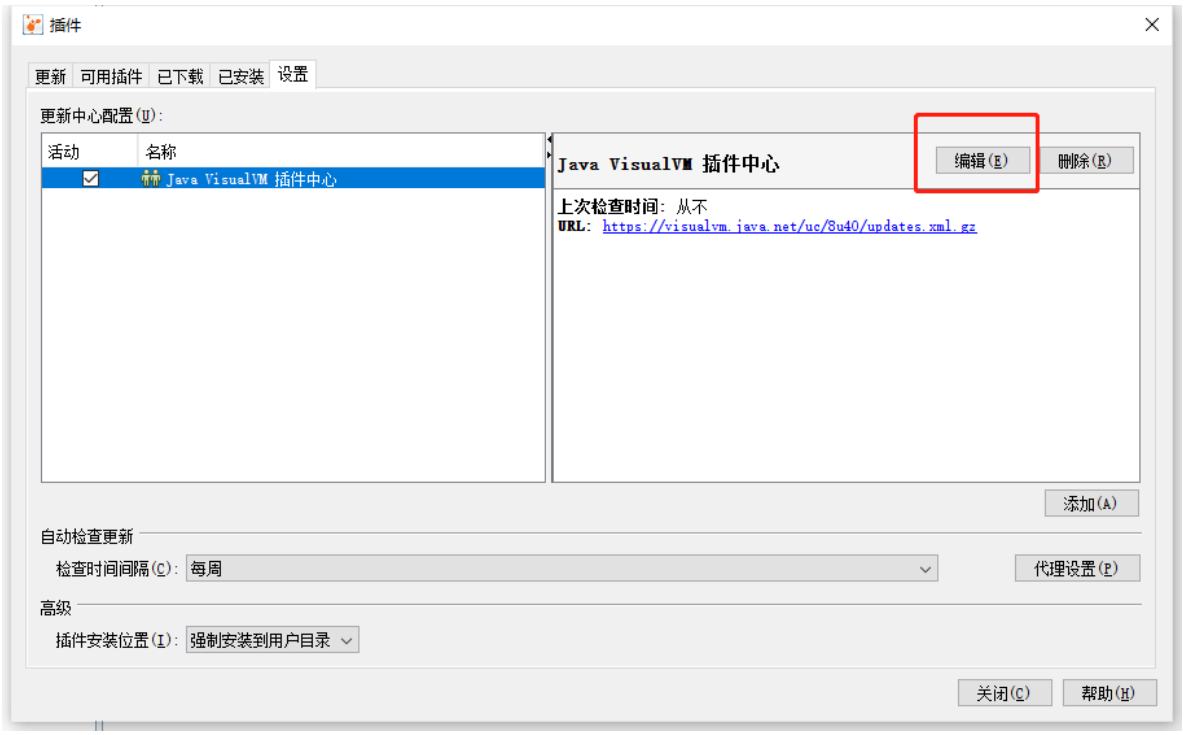
我是1.8的 所以选择这个

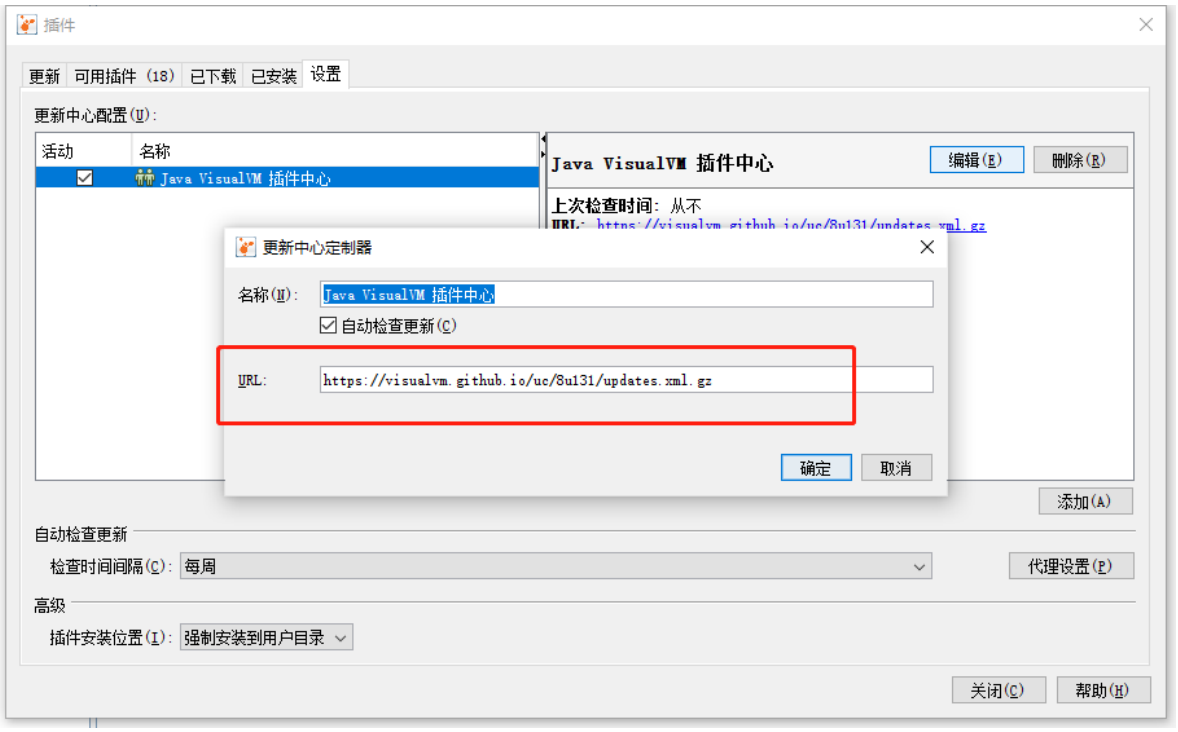


VisualVM	Java VisualVM
VisualVM 2.1.2 https://visualvm.github.io/uc/release212/updates.xml.gz	**JDK 8 Update 131 - 321** https://visualvm.github.io/uc/8u131/updates.xml.gz
VisualVM 2.1.1 https://visualvm.github.io/uc/release211/updates.xml.gz	**JDK 8 Update 40 - 121** **JDK 7 Update 79 - 80** https://visualvm.github.io/archive/uc/8u40/updates.xml.gz
VisualVM 2.1 https://visualvm.github.io/uc/release21/updates.xml.gz	**JDK 8 Update 20 - 25** https://visualvm.github.io/archive/uc/8u20/updates.xml.gz
VisualVM 2.0.7 https://visualvm.github.io/uc/release207/updates.xml.gz	**JDK 8 - Update 11** **JDK 7 Update 60 - 76** https://visualvm.github.io/archive/uc/7u60/updates.xml.gz
VisualVM 2.0.6 https://visualvm.github.io/uc/release206/updates.xml.gz	**JDK 7 Update 40 - 55** https://visualvm.github.io/archive/uc/7u14/updates.xml.gz
VisualVM 2.0.5 https://visualvm.github.io/uc/release205/updates.xml.gz	**JDK 7 Update 6 - 25** https://visualvm.github.io/archive/uc/7u6/updates.xml.gz
VisualVM 2.0.4 https://visualvm.github.io/uc/release204/updates.xml.gz	**JDK 7 Update 4 - 5** https://visualvm.github.io/archive/uc/7/updates.xml.gz
VisualVM 2.0.3 https://visualvm.github.io/uc/release203/updates.xml.gz	**JDK 7 - Update 3** **JDK 6 Update 30 - 45** https://visualvm.github.io/archive/uc/7/updates.xml.gz
VisualVM 2.0.2 https://visualvm.github.io/uc/release202/updates.xml.gz	**JDK 6 Update 23 - 29** https://visualvm.github.io/archive/uc/6u23/updates.xml.gz
VisualVM 2.0.1 https://visualvm.github.io/uc/release201/updates.xml.gz	**JDK 6 Update 21 - 22** https://visualvm.github.io/archive/uc/6u20/updates.xml.gz
VisualVM 2.0 https://visualvm.github.io/uc/release20/updates.xml.gz	**JDK 6 Update 18 - 20** https://visualvm.github.io/archive/uc/6u18/updates.xml.gz
VisualVM 1.4.4 https://visualvm.github.io/uc/release144/updates.xml.gz	**JDK 6 Update 14 - 17**
VisualVM 1.4.3	

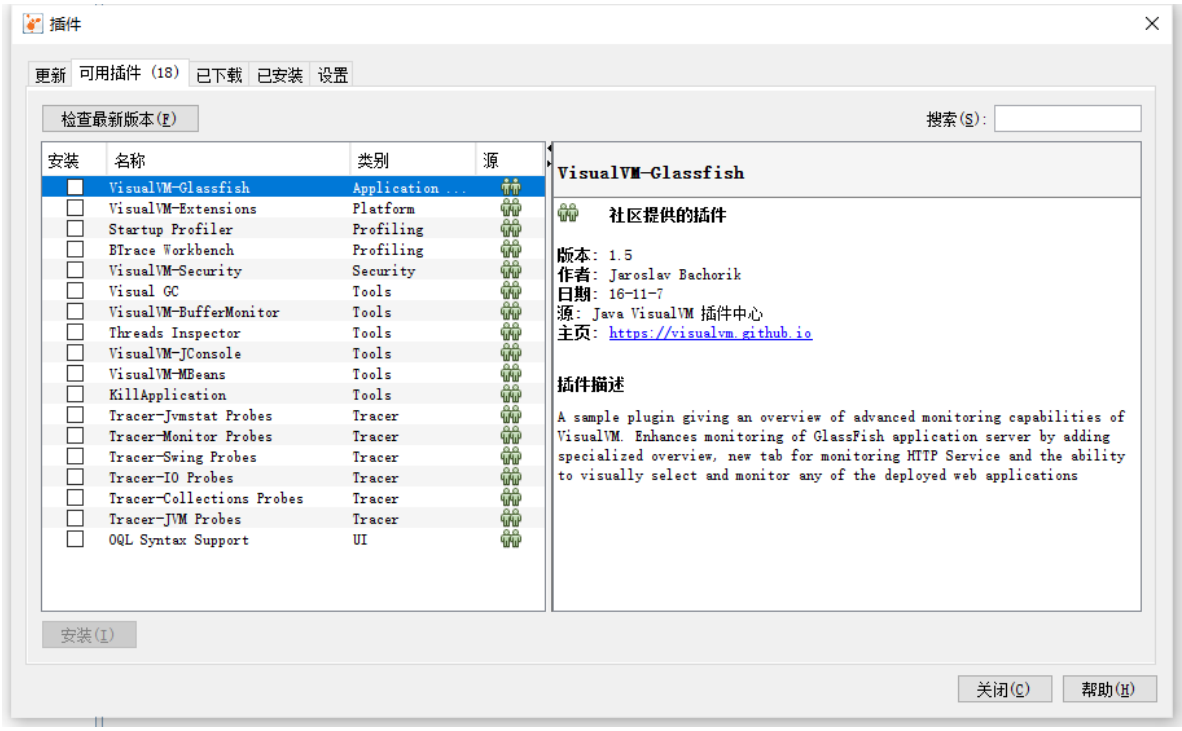
复制对应的链接：https://visualvm.github.io/uc/8u131/updates.xml.gz

然后修改URL



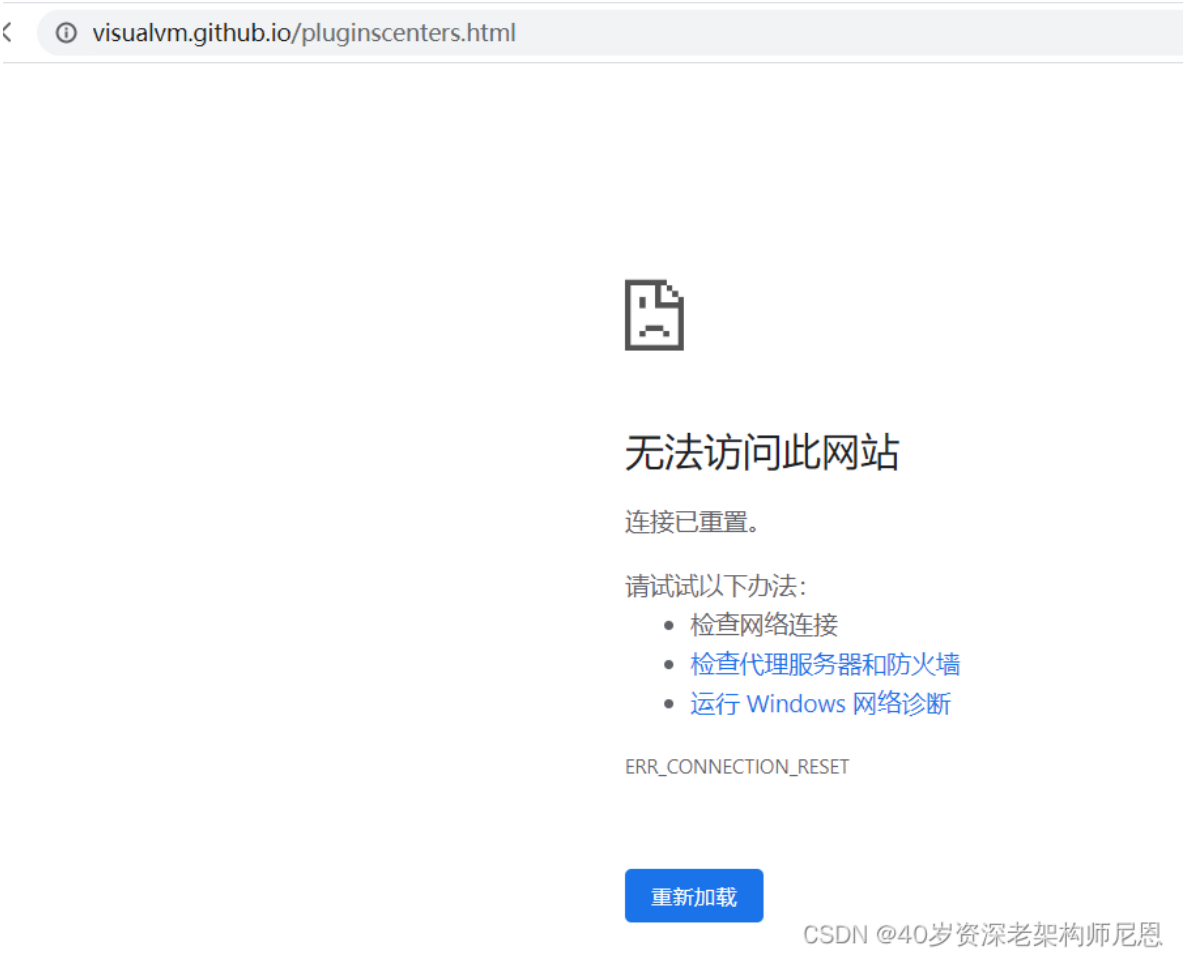


然后就可以看到插件列表了

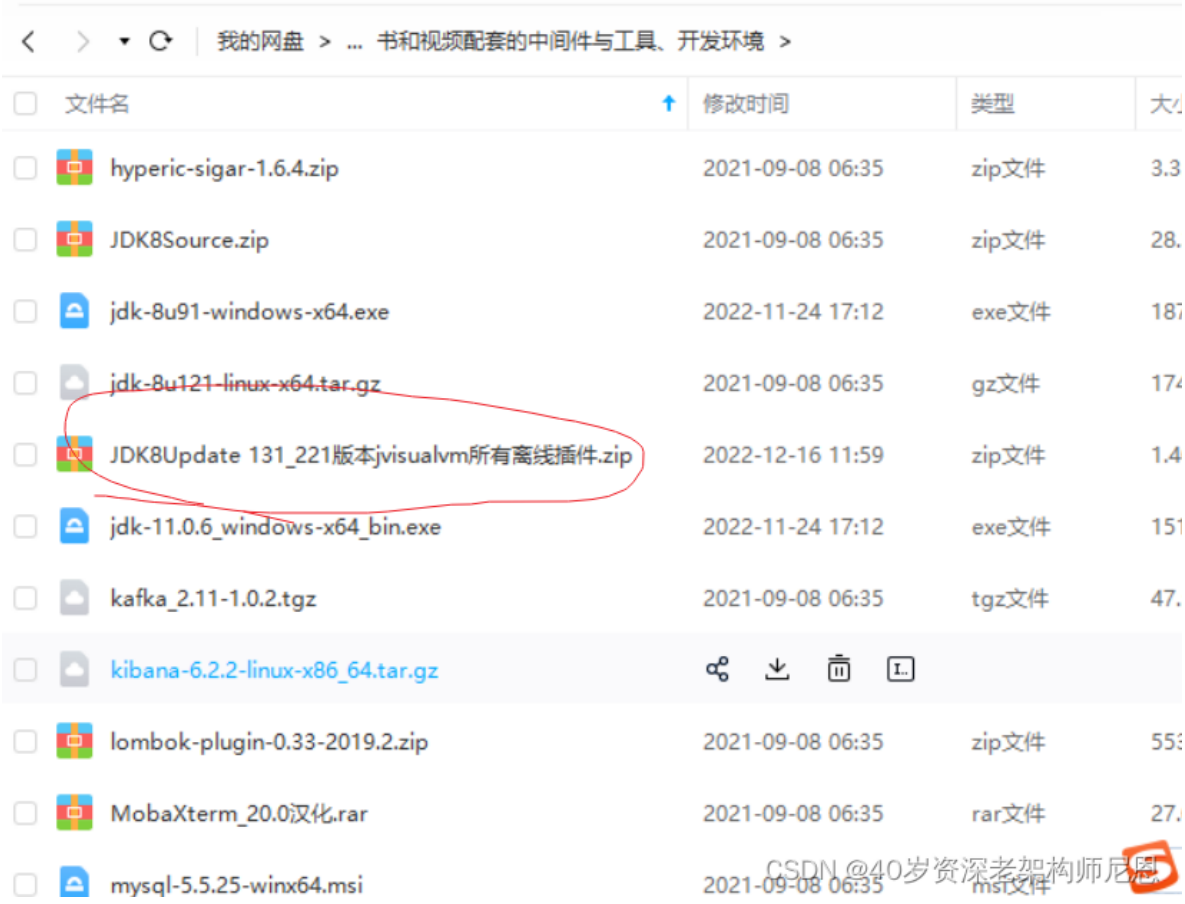


离线下载插件包

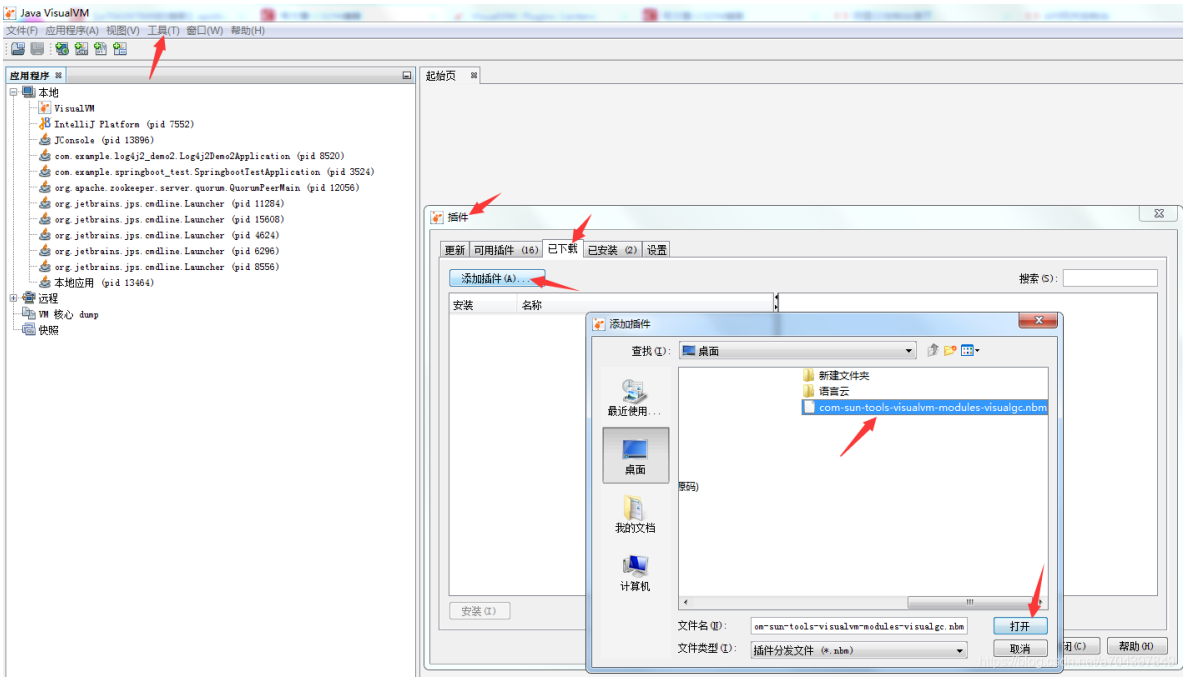
在线地址，不能开启， 在线更新 插件地址， 没有路了



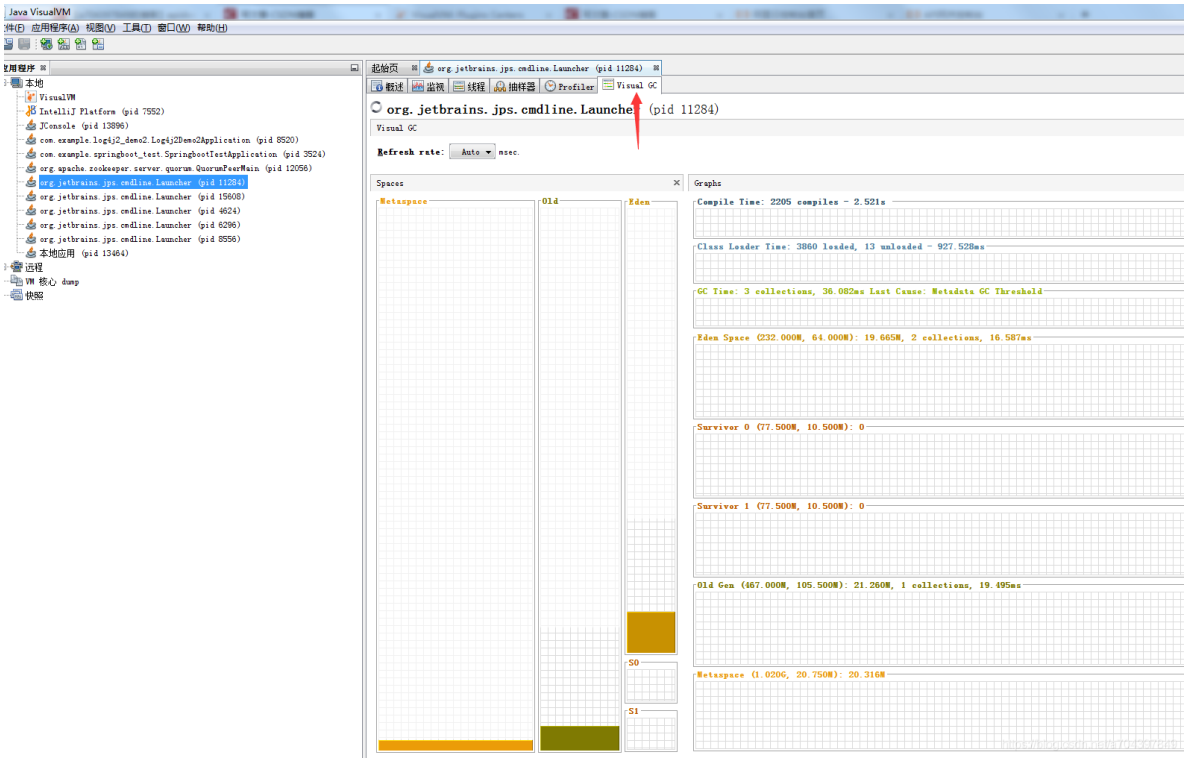
离线下载插件包



打开jdk安装后bin目录下的jvisualvm -> 工具 -> 插件 -> 已下载 -> 添加插件 -> 打开下载好的离线插件 -> 安装



安装完后 就可以看到监控信息 多了个 Visual GC (如果安装完了没有，尝试重启jvisualvm)



注意：如果打开 Visual GC 工具，提示:不受此JVM 支持，

那是因为远程服务器上需要启动jstatd代理程序。

打开本地java应用jvm连接的Visual GC 可以看到内存信息，是因为本地jvisualvm 默认启动了jstatd代理程序

解决网络问题：

visualvm.github.io

https://visualvm.github.io/pluginscenters.html

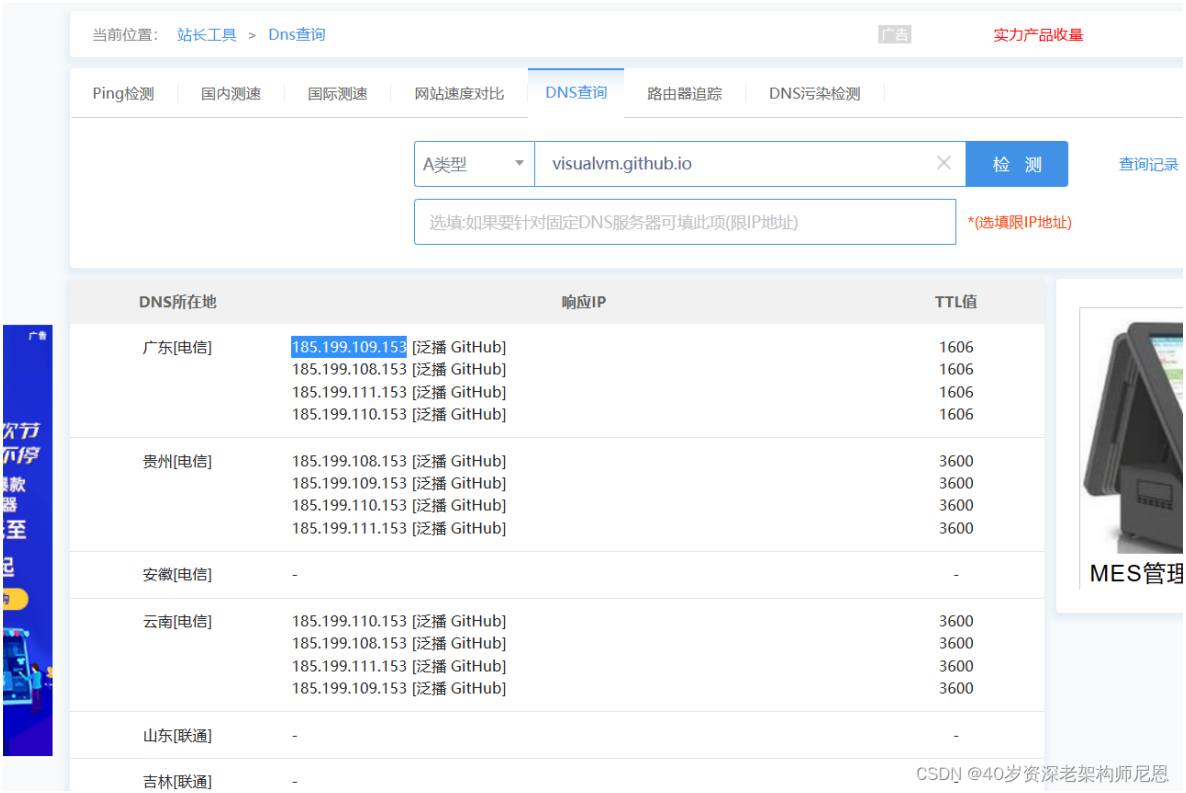
dns域名不能解析

出现原因是 商的dns服务器有问题，所以改成了外部dns服务器，114.114.114.114，然后就可以访问了。

测试后无效

自己找IP

最后一个办法了。
http://tool.chinaz.com/dns



如果没有IP，多刷几次就可以了。

然后改hosts文件，需要用管理员权限才可以生效。

```
185.199.109.153  visualvm.github.io
```

https://185.199.111.153/pluginscenters.html

https://visualvm.github.io/pluginscenters.html

实战内容:

实操1：OOM后系统已挂，使用JVisualVM分析内存溢出OOM

前置条件:

oom时导出了堆的dump文件

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=

分析路径:

- 查看占用 内存多的对象
- 找到对象实例的GCRoot 对象
- 查看线程栈、找到业务代码

实操2：内存泄漏，使用JVisualVM分析内存泄漏

分析路径：

系统运行中、内存消耗越来越大

对比 dump文件，看看那些对象实例， 是不断增加的， 就有可能存在内存溢出

实操1：使用JVisualVM分析内存溢出OOM

OOM后系统已挂，使用JVisualVM分析内存溢出OOM

分析路径：

- 查看占用 内存多的对象
- 找到对象实例的GCRoot 对象
- 查看线程栈、找到业务代码

前置条件：

oom时导出了堆的dump文件

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=

分析路径：

- 查看占用 内存多的对象
- 找到GCRoot
- 查看线程栈、找到业务代码

前提：oom时导出了堆的dump文件

HeapDumpOnOutOfMemoryError使用方法

1、配置方法

在JAVA_OPTIONS变量中增加

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=${目录}。
```

例如：

```
export JAVA_OPTS="-Xms2048M -Xmx2048M -Xmn682M -XX:MaxPermSize=96M"
```

2、参数说明

（1）-XX:+HeapDumpOnOutOfMemoryError参数表示当JVM发生OOM时，自动生成DUMP文件。

（2）-XX:HeapDumpPath=${目录}参数表示生成 DUMP文件的路径，也可以指定文件名称，

例如：

```
-XX:HeapDumpPath=${目录}/java_heapdump.hprof
```

如果不指定文件名，默认为：

```
java_<pid>_<date>_<time>_heapDump.hprof
```

HeapDumpPath

使用-XX:HeapDumpPath配置的时候，需要保证目录的文件夹都是存在，

因为它在到处dump文件的时候，不会帮你去创建不存在的目录。

使用相对路径：

```
 nohup java -Xms512M -Xmx512M -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=./  -jar xxx.jar  >/dev/null 2>&1 &
```

使用绝对路径：

```
 nohup java -Xms512M -Xmx512M -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/vagrant/chapter26/  -jar xxx.jar  >/dev/null 2>&1 &
```

准备模拟内存泄漏demo

1、定义静态变量HashMap

2、分段循环创建对象，并加入HashMap

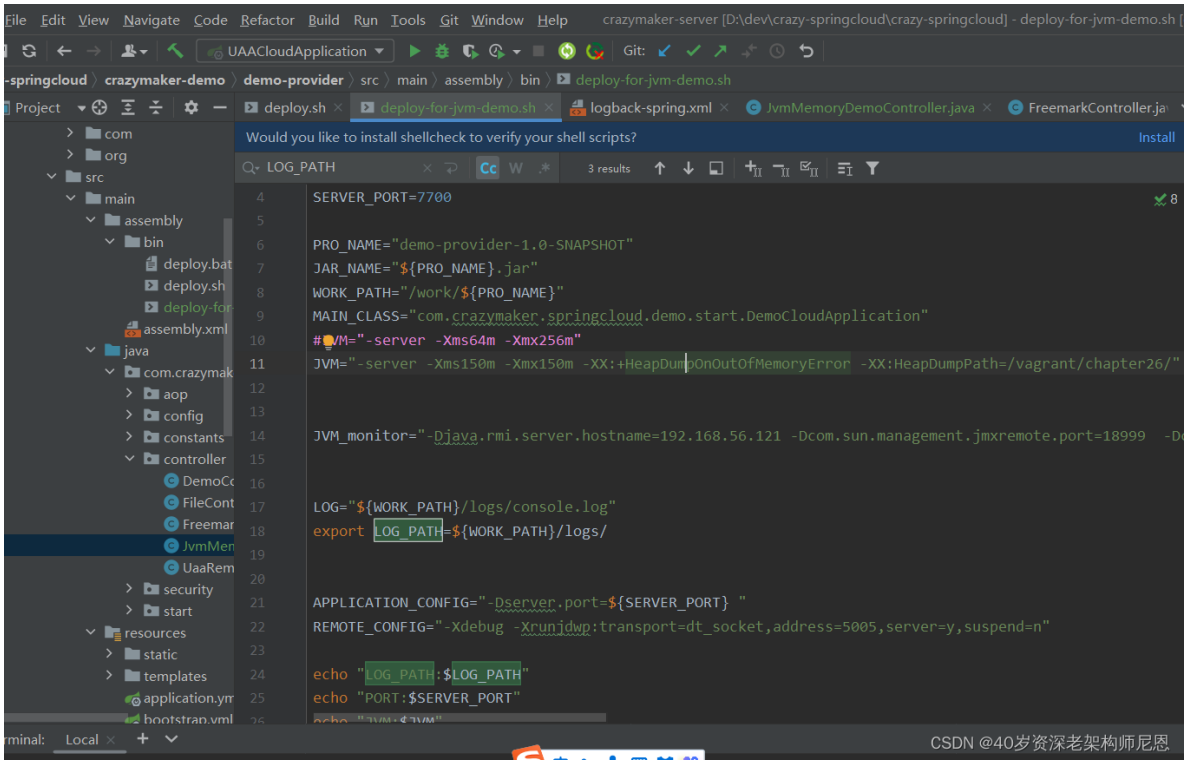
代码如下：

```
package com.crazymaker.springcloud.demo.controller;

import com.alibaba.fastjson.JSONObject;
import com.crazymaker.springcloud.common.result.RestOut;
import com.crazymaker.springcloud.common.util.ThreadUtil;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import java.util.HashMap;
import java.util.Map;


@Api(value = "JvmMemoryDemo", tags = {"JvmMemory"})

@RestController
@RequestMapping("/jvm/file")
public class JvmMemoryDemoController {

    private int count=0;

    static class TestMemory {
        int foo;
    }

    //声明缓存对象
    private static final Map map = new HashMap();


    @GetMapping("/addObject/v1")
    @ApiOperation(value = "添加对象到缓存")
    public RestOut<JSONObject> addObject()
    {

        //循环添加对象到缓存
        for(int i=0; i<1_000_000;i++){
            TestMemory t = new TestMemory();
            map.put("key"+i,t);
        }
        JSONObject data = new JSONObject();

        data.put("第N次操作：", ++count);
        return RestOut.success(data).setRespMsg("操作成功");

    }

}
```

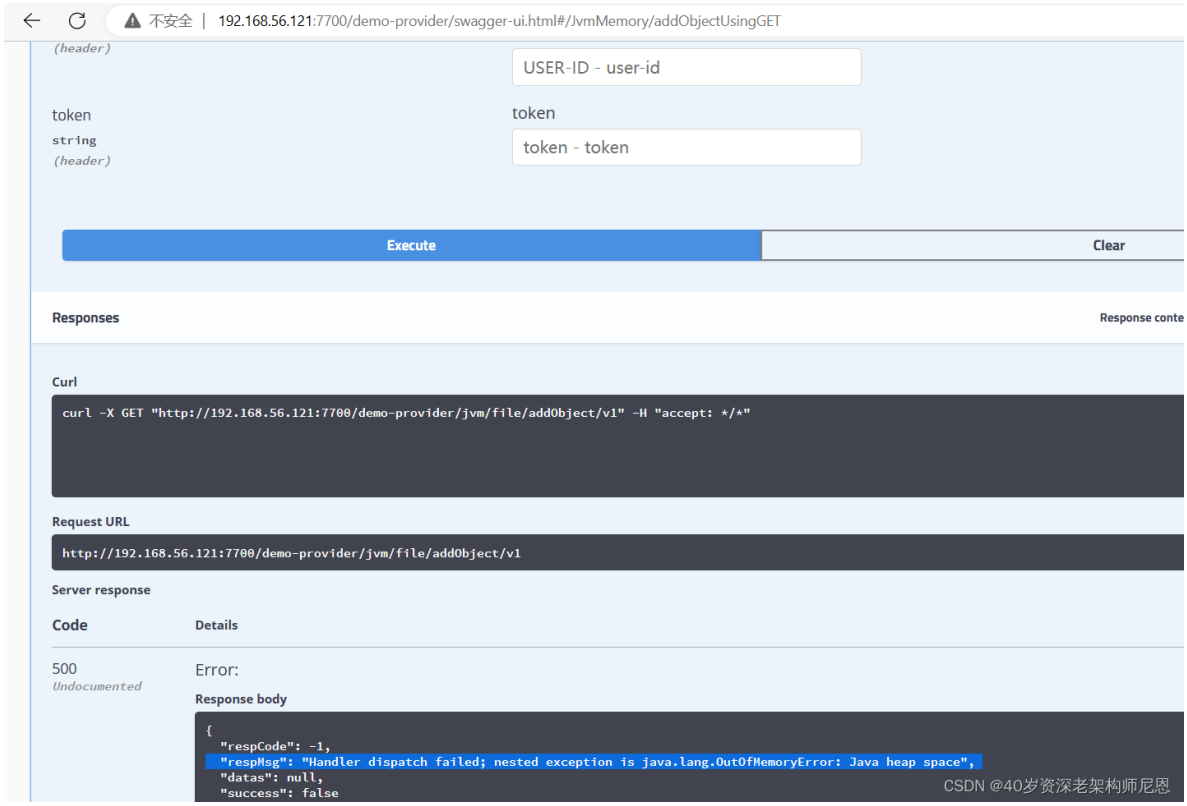
3、配置jvm参数如下：

```
         -Xms150m
         -Xmx150m
```



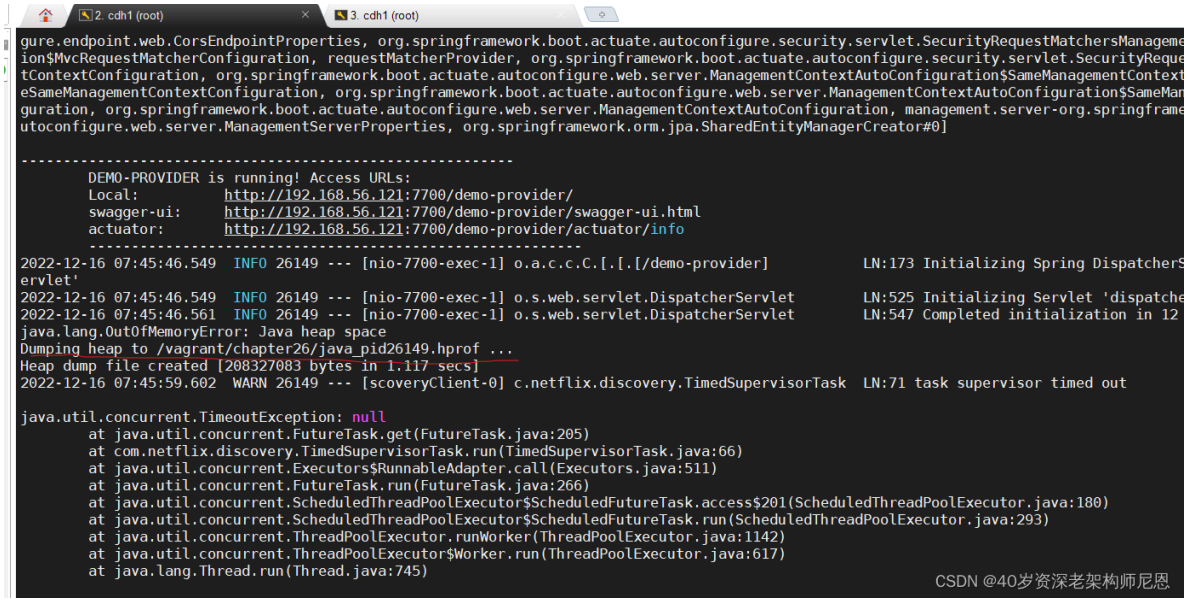
第1步： 点击 测试接口

http://192.168.56.121:7700/demo-provider/swagger-ui.html

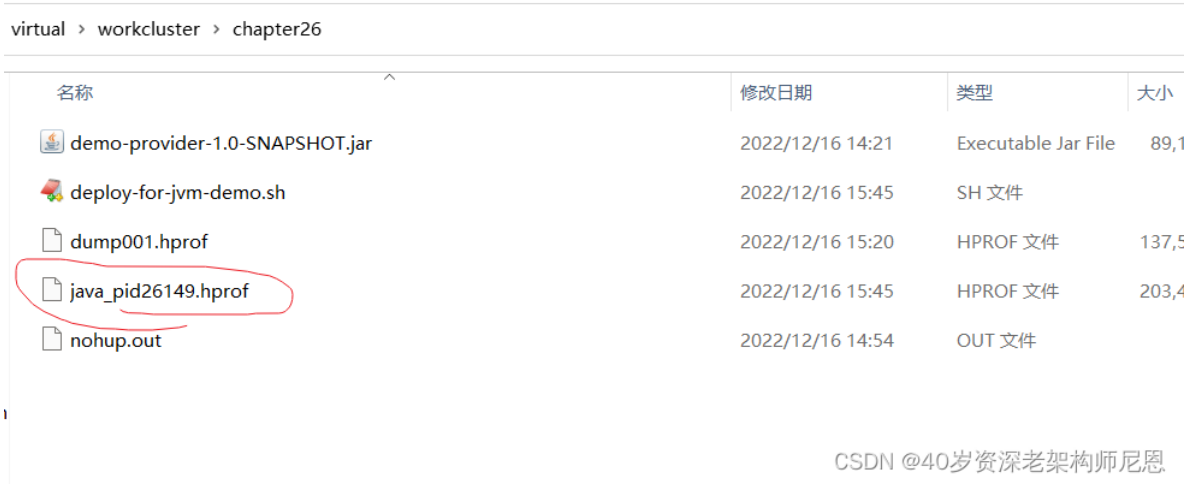


第2步： Jvm发生内存溢出，再说导出堆

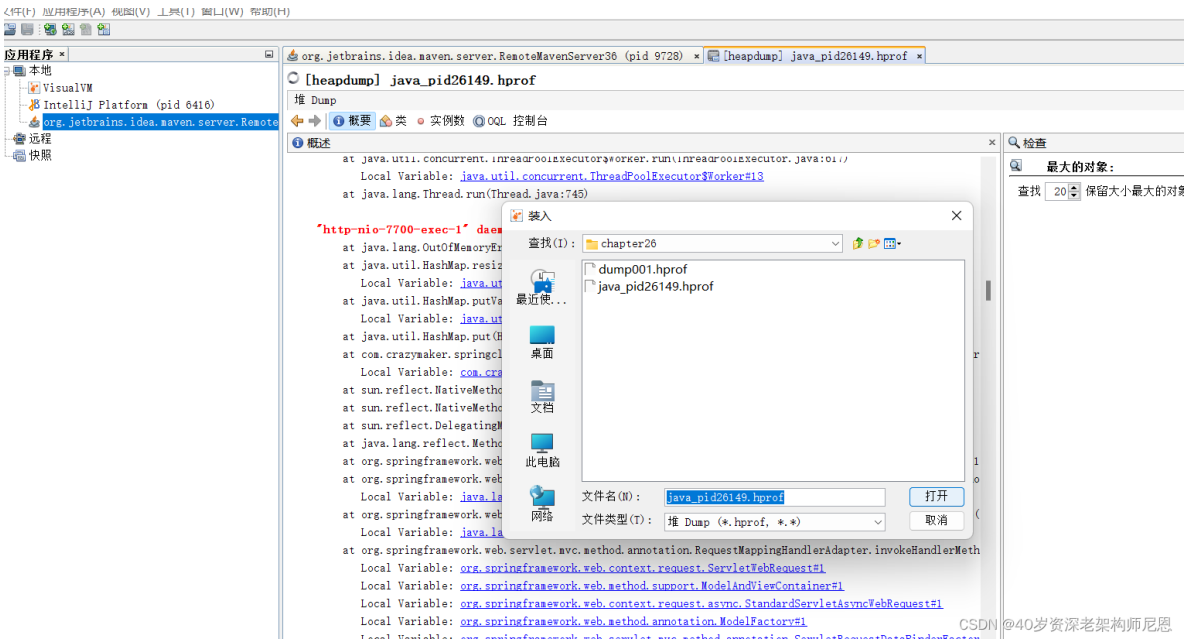
```
java.lang.OutOfMemoryError: Java heap space
Dumping heap to /vagrant/chapter26/java_pid26149.hprof ...
Heap dump file created [208327083 bytes in 1.117 secs]
```



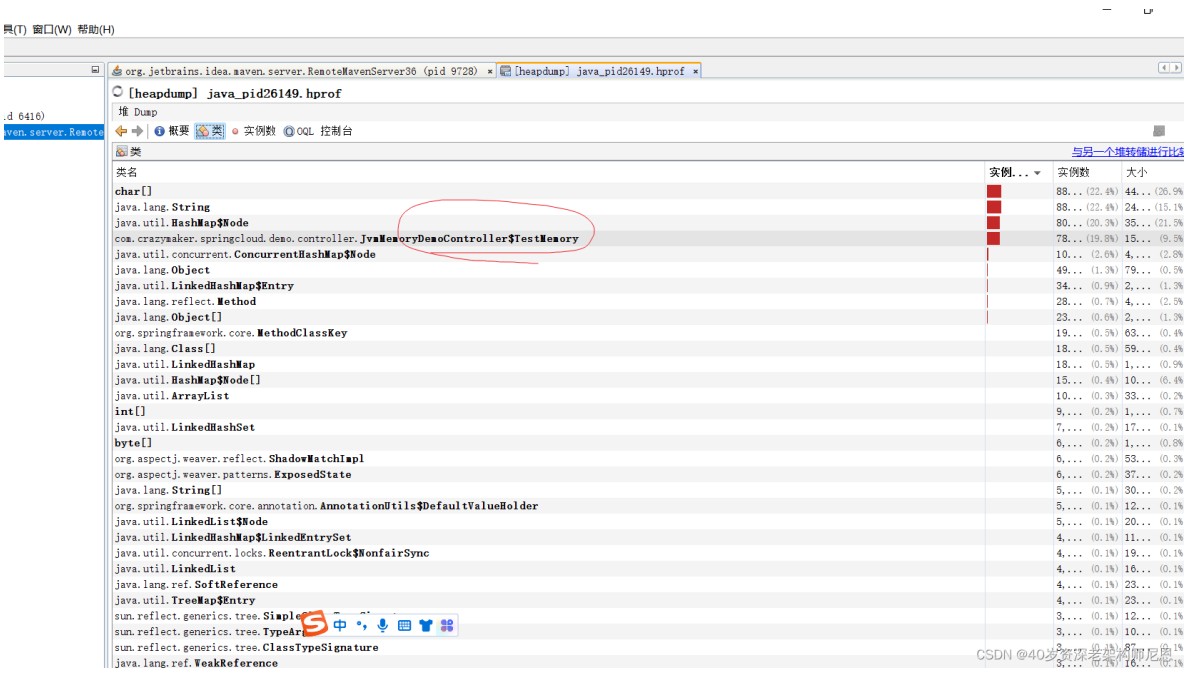
第3步： 找到dump文件



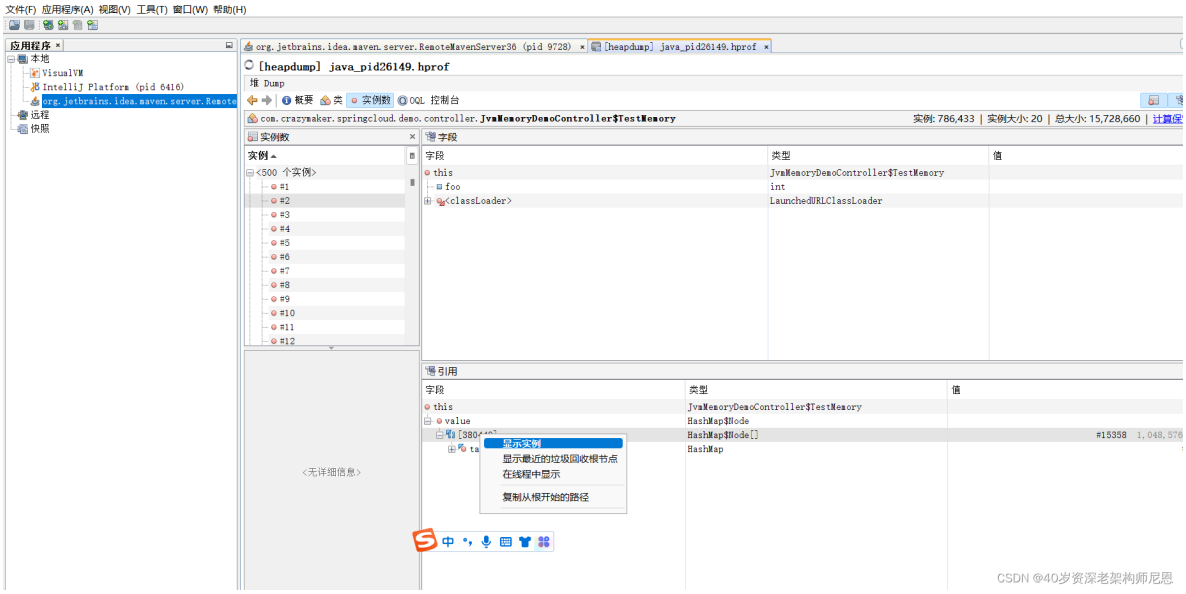
第4步： 装入dump文件到JvirtualVm



第5步： JVisualVM查看类的信息

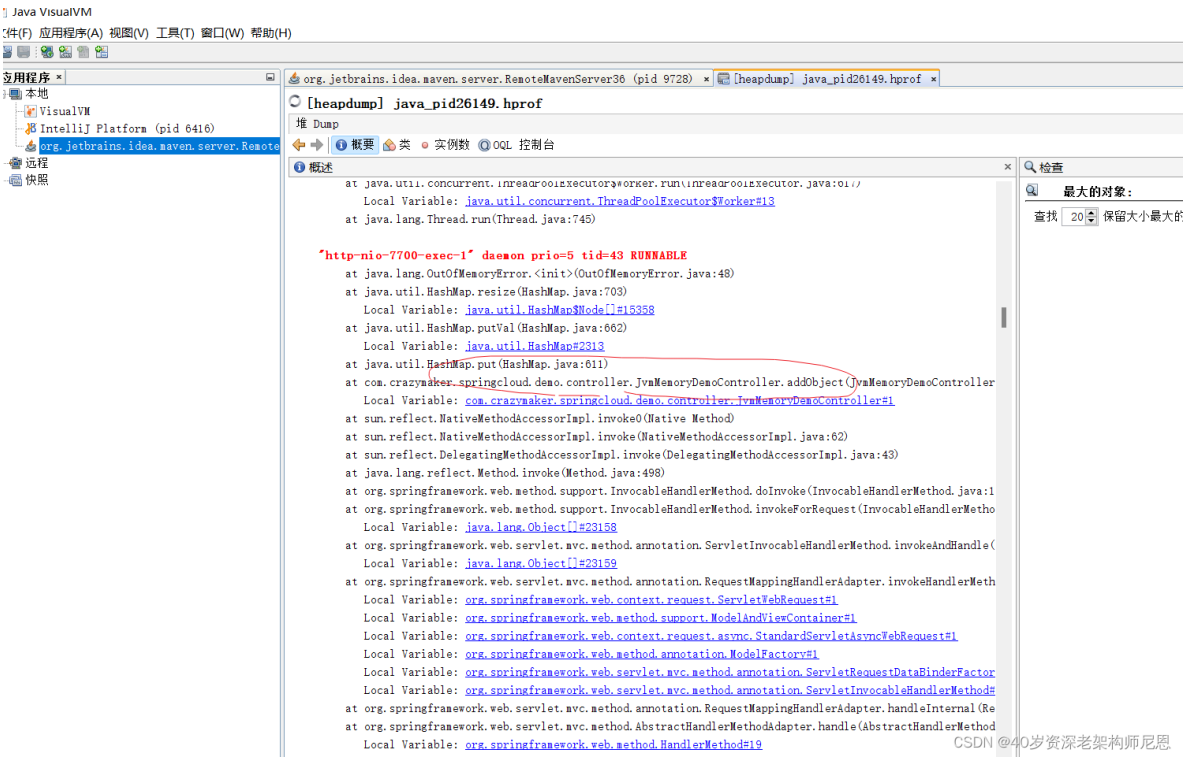


第6步： JVisualVM查看实例的信息



通过GC-Root 对象，查看线程信息

第7步： JVisualVM查看线程的信息



实操2：使用JVisualVM分析内存泄漏

分析路径：

系统运行中、内存消耗越来越大

对比 dump文件，看看那些对象实例， 是不断增加的， 就有可能存在内存溢出

相关命令：导出堆的dump文件

1.查看进程id

jps

2.查看内存状态

jmap -heap 进程ID

3.查看JVM堆中对象详情占用情况
jmap -histo 进程ID

4.导出整个JVM 中内存信息，可以利用其它工具打开dump文件分析，例如jdk自带的visualvm工具

jmap -dump:file=文件名.dump [pid]

jmap -dump:format=b,file=文件名 [pid]

format=b指定为二进制格式文件

一般情况下，是 在shell脚本，配置这些选项：

使用脚本，速度更快：

```
function dump() {
    pid=$(ps -ef | grep -v 'grep' | egrep $JAR_NAME| awk '{printf $2 " "}')
    jps
    echo "${JAR_NAME} is running and pid is $pid"
    if [ "$pid" != "" ]; then
#    jmap -dump:format=b,file=文件名
     cmd="jmap -dump:format=b,file=dump001.hprof  $pid"
     echo $cmd
     eval $cmd

     ls -l
    else
        echo "${JAR_NAME} is stopped"
    fi
    status
}
```

第1步：调整配置

免得一次就oom了，这一次，需要晚点oom

```
PRO_NAME="demo-provider-1.0-SNAPSHOT"
JAR_NAME="${PRO_NAME}.jar"
WORK_PATH="/work/${PRO_NAME}"
MAIN_CLASS="com.crazymaker.springcloud.demo.start.DemoCloudApplication"
#JVM="-server -Xms64m -Xmx256m"
#JVM="-server -Xms150m -Xmx150m -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/vagrant/chapter26/"
JVM="-server -Xms500m -Xmx4G -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/vagrant/chapter26/"
```

第2步：开始启动应用

http://192.168.56.121:7700/demo-provider/swagger-ui.html

第3步：JVisualVM 远程监控 SpringBoot应用

1、修改远程jvm的启动命令，在其中增加：

```
JVM_monitor="-Djava.rmi.server.hostname=192.168.56.121 -
Dcom.sun.management.jmxremote.port=18999  -
Dcom.sun.management.jmxremote.rmi.port=18998 -Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.local.only=false  -
Dcom.sun.management.jmxremote.authenticate=false  -
Dcom.sun.management.jmxremote.ssl=false"
```

这次配置先不走权限校验。

只是打开jmx端口。

2、打开jvisualvm，右键远程，选择添加远程主机：

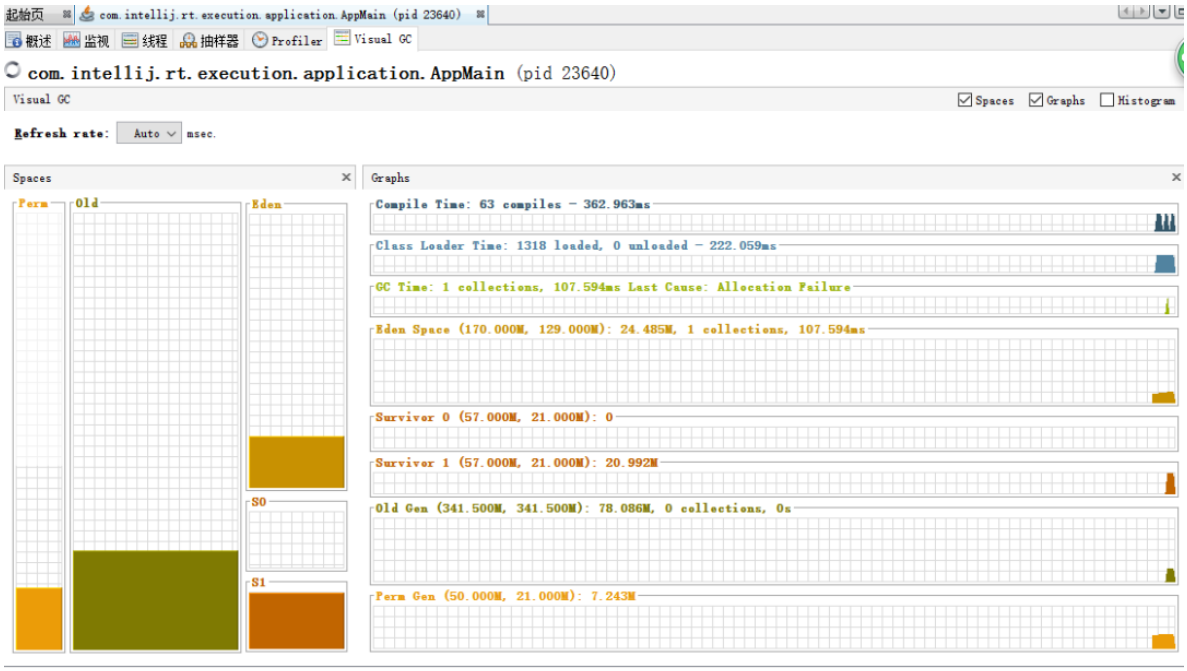
3、输入主机的名称，直接写ip，如下：

右键新建的主机，选择添加JMX连接，输入在tomcat中配置的端口即可。

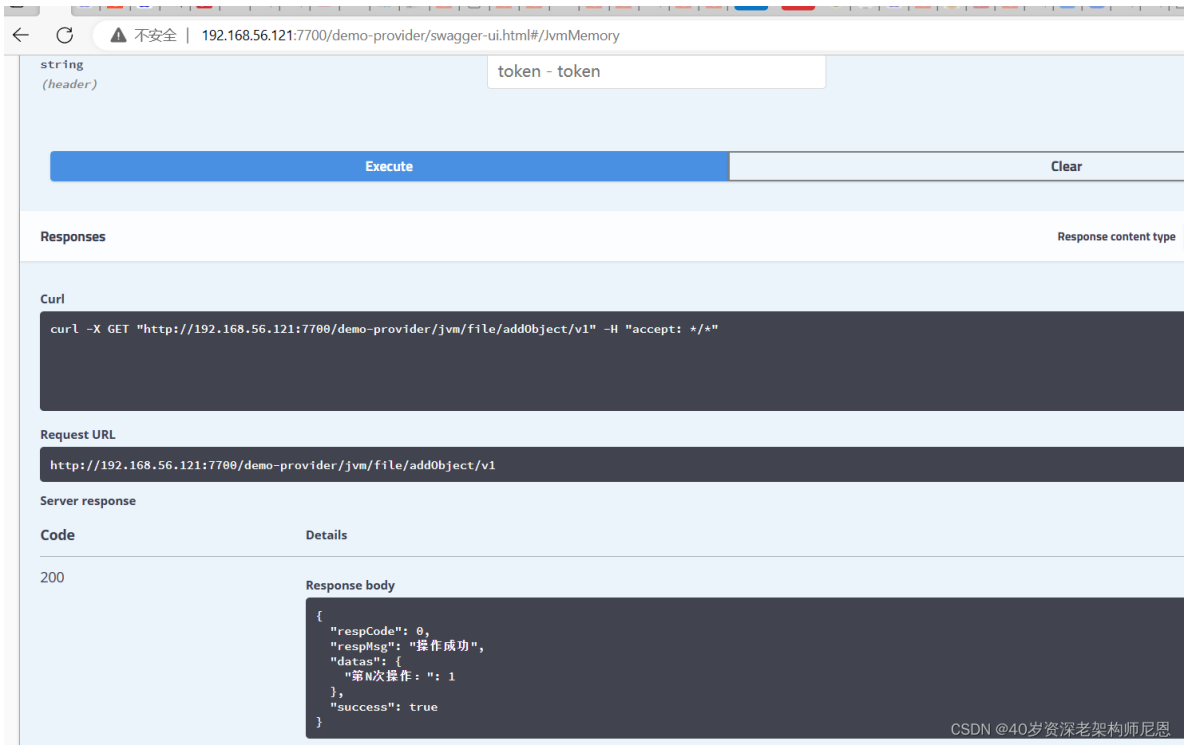
4、双击打开。

导入JVisualVM，并且Visual GC标签，内容如下，

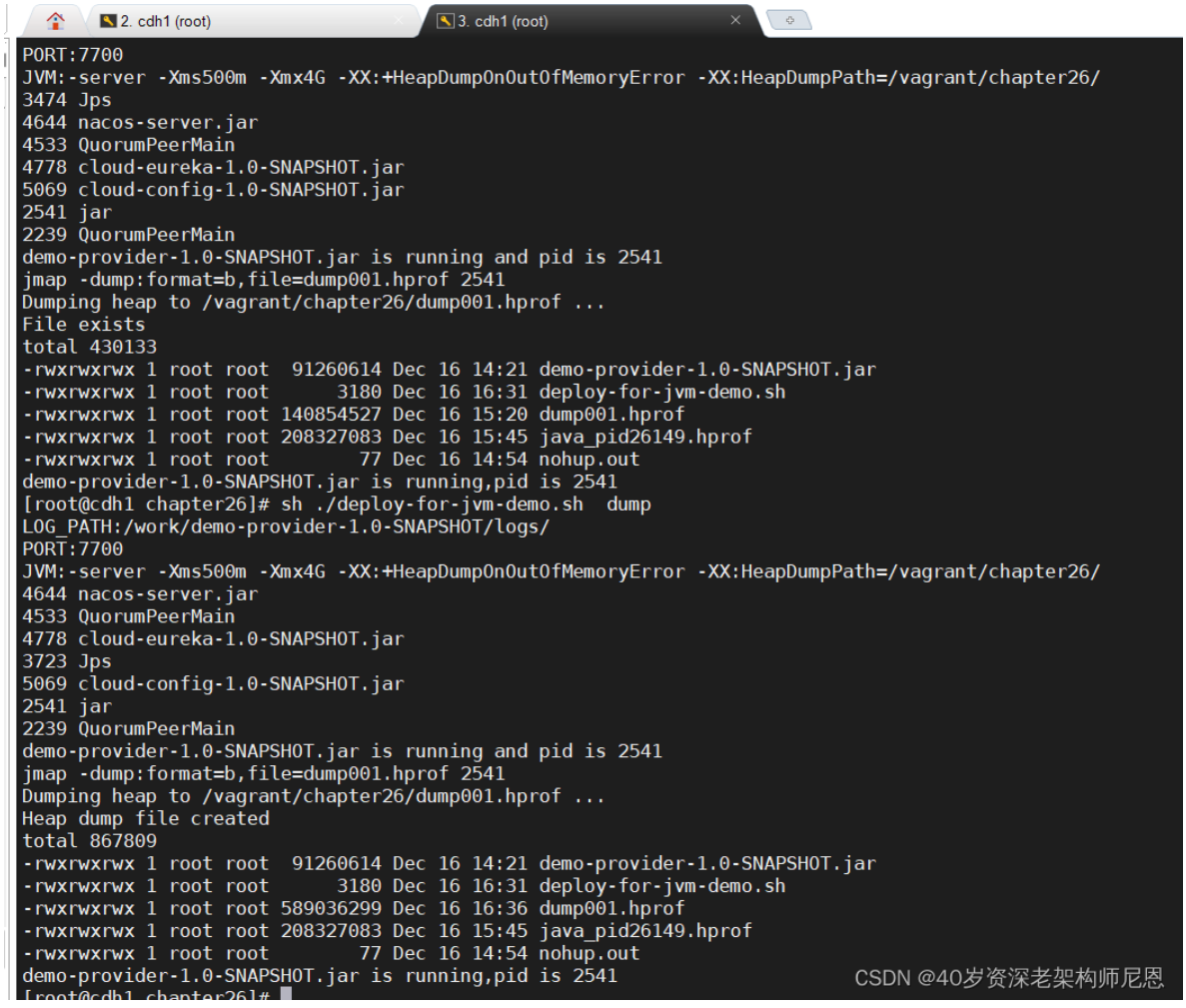
这是输出first的截图



第4步：第一次请求和dump



dump 堆，这里使用 脚本，没有直接使用 jmap 命令

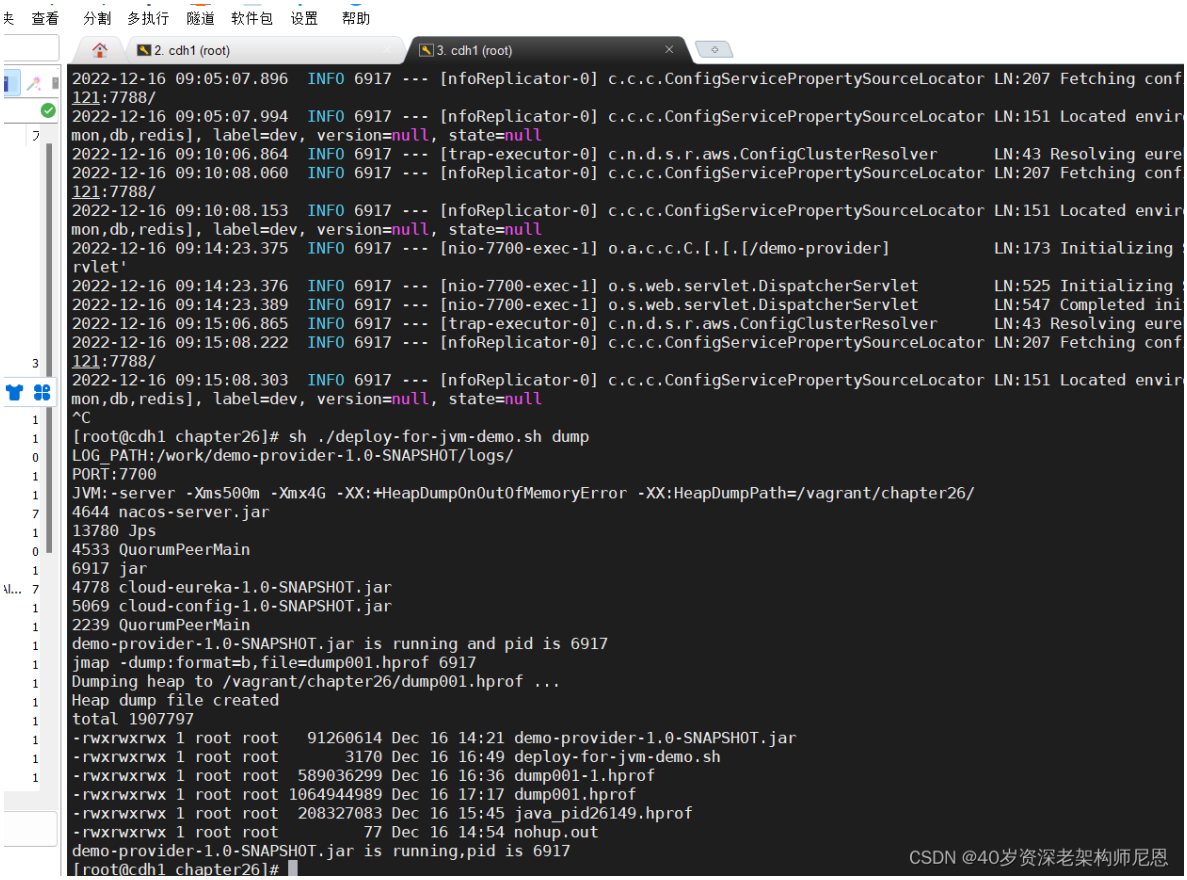


第1次请求之后，

第5步：第2次请求和dump

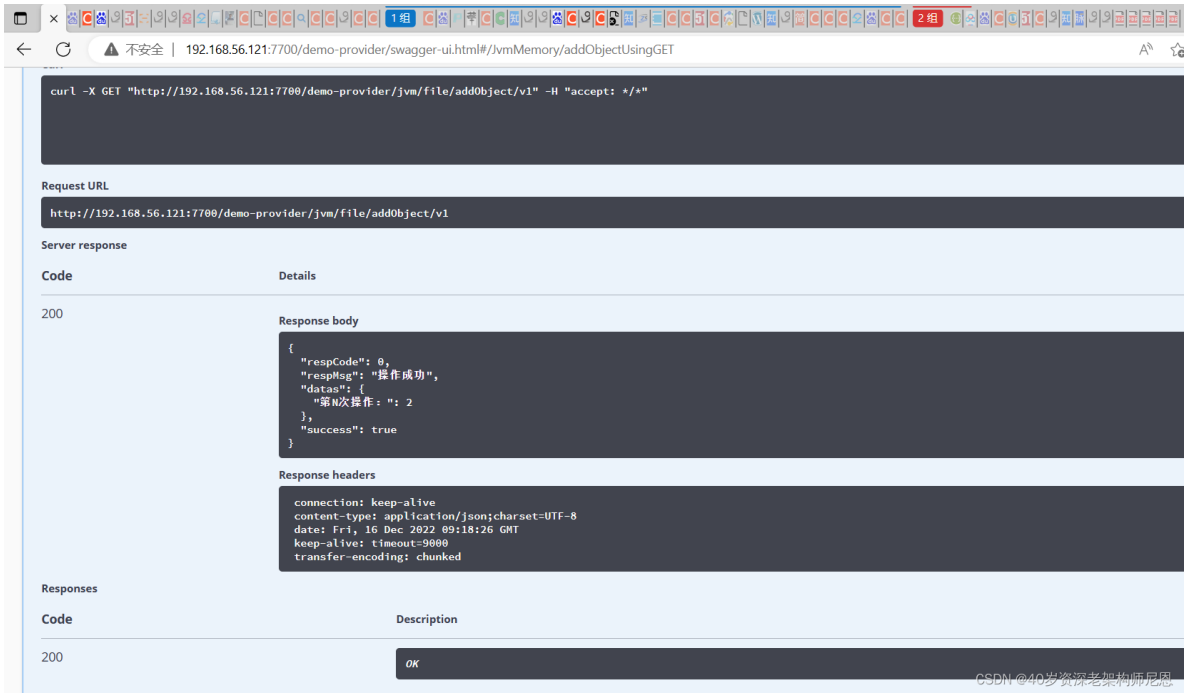
第2次请求之后

dump 堆，这里使用 脚本，没有直接使用 jmap 命令

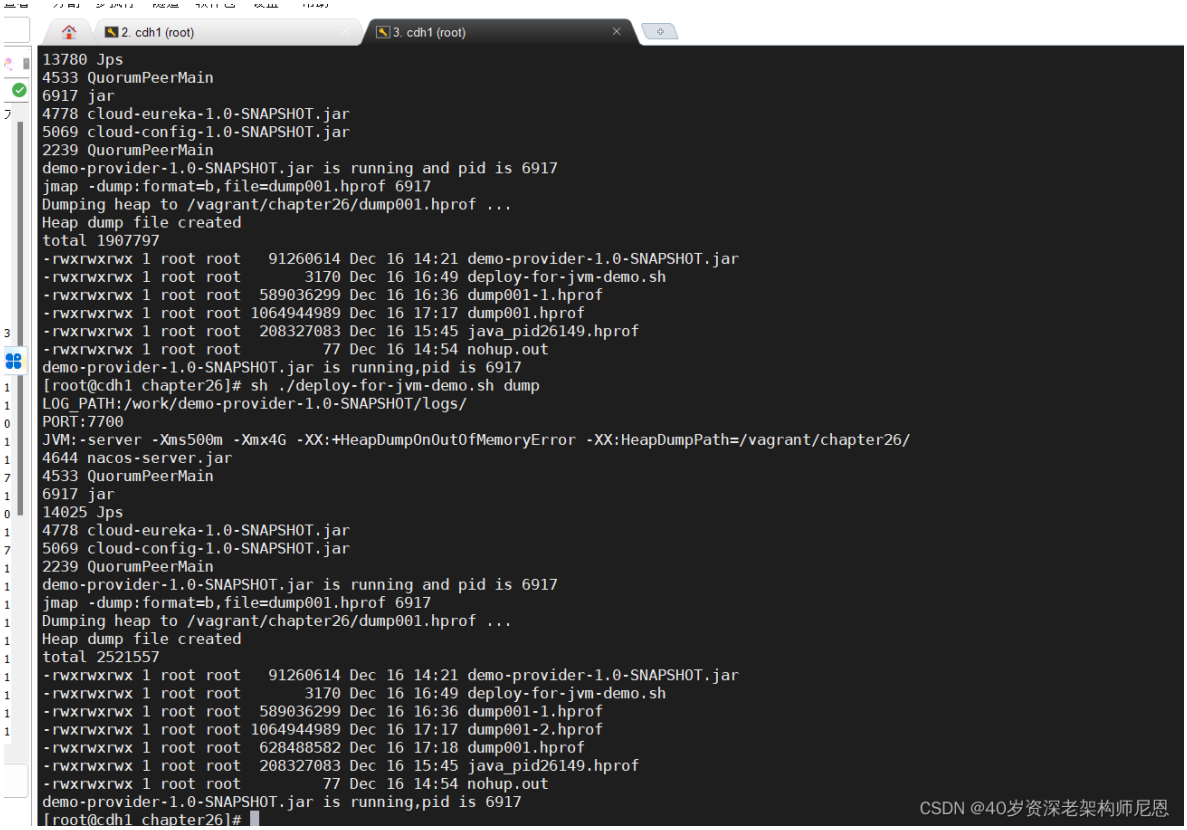


第6步：第6次请求和dump

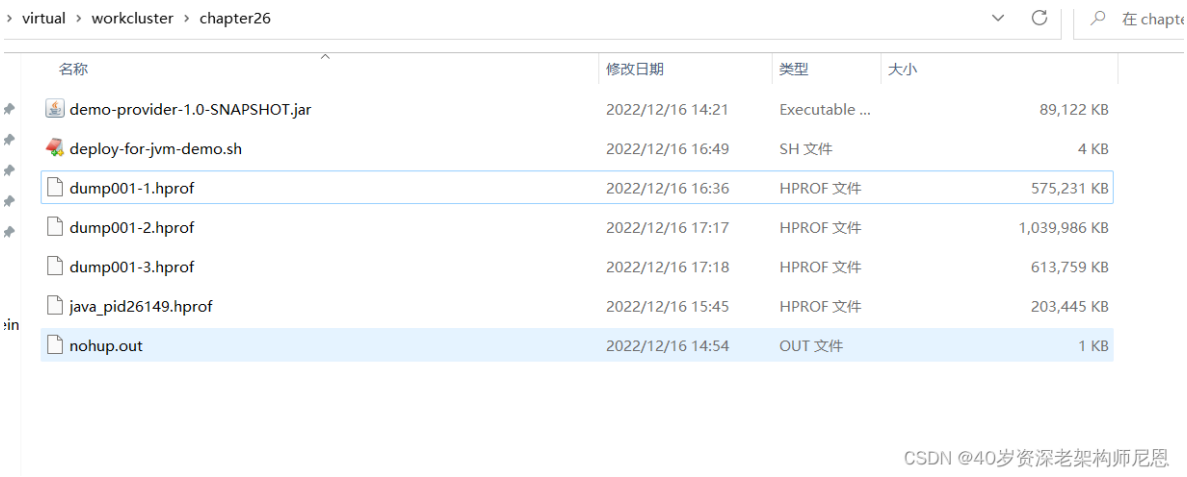
第6次请求之后



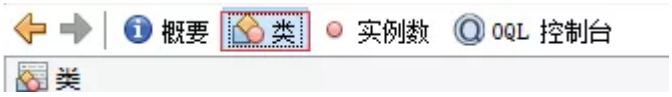
dump 堆，这里使用 脚本，没有直接使用 jmap 命令



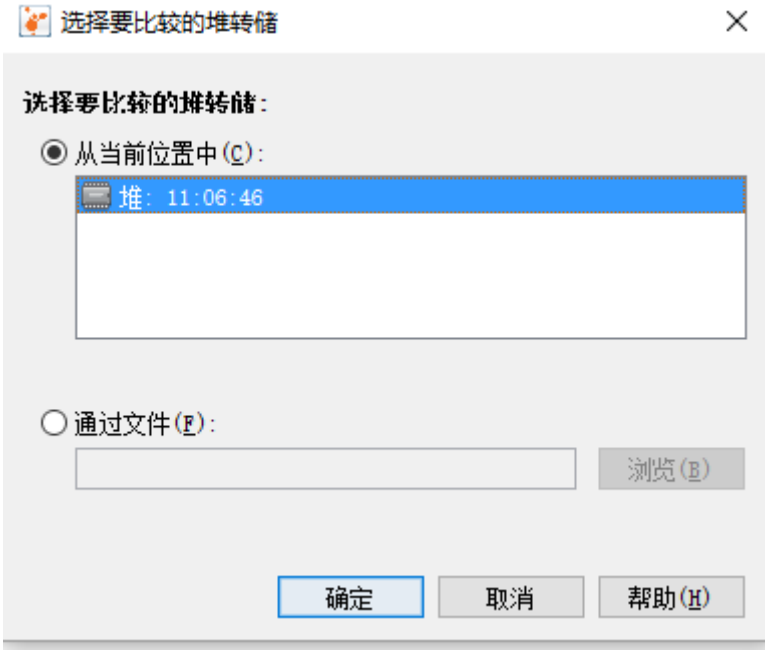
第3步：对照三个dump文件

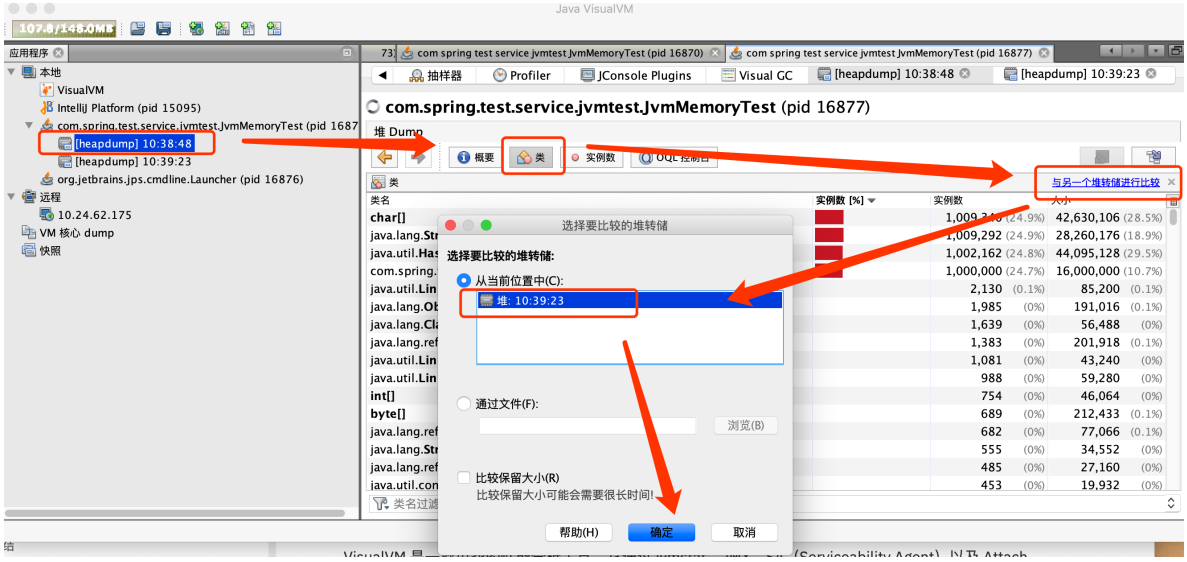


进入最后dump出来的堆标签，点击类：



点击右上角："与另一个堆存储对比"。如图选择第一次导出的dump内容比较：

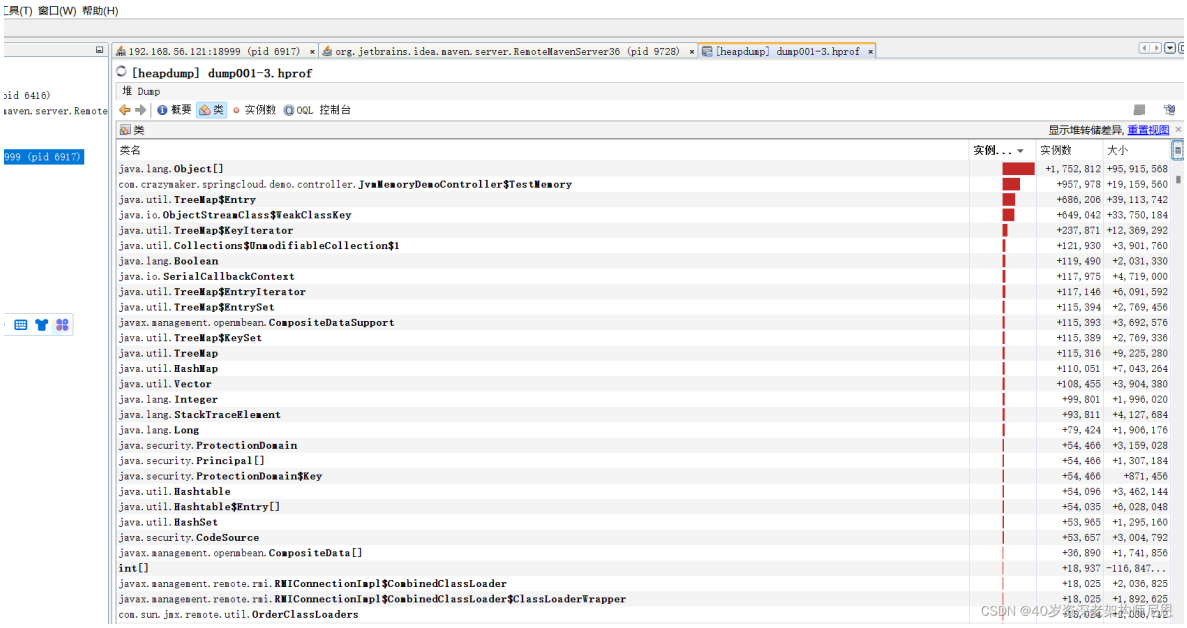




比较结果如下:

类

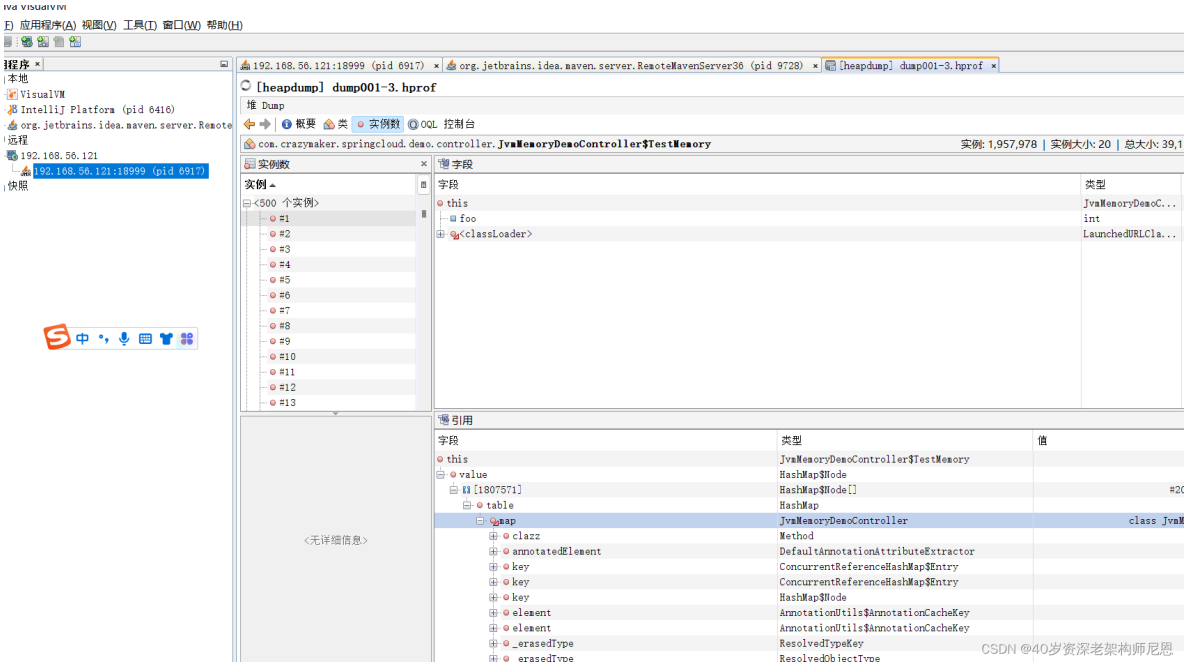
类名	实例数 [%]	实例数
char[]		+3,000,117
java.lang.String		+3,000,117
TestMemory		+3,000,000
java.util.HashMap$Entry		+3,000,000



可以看出在两次间隔时间内TestMemory对象实例一直在增加并且多了，说明该对象引用的方法可能存在内存泄漏。

如何查看对象引用关系呢?

右键选择类TestMemory，选择“在实例视图中显示”，如下所示:



- 左侧是创建的实例总数，
- 右侧上部为该实例的结构，
- 下面为引用说明，

从图中可以看出在类JvmMemoryDemoController里面被引用了，并且被HashMap引用。

如此可以确定泄漏的位置，进而根据实际情况进行分析解决。

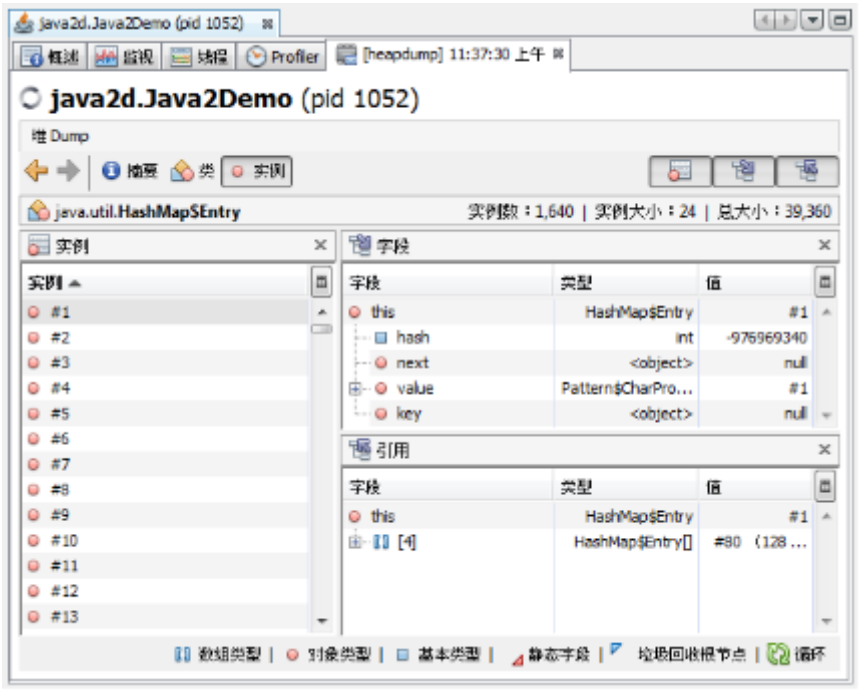
如何进行JVM调优

观察内存释放情况、集合类检查、对象树

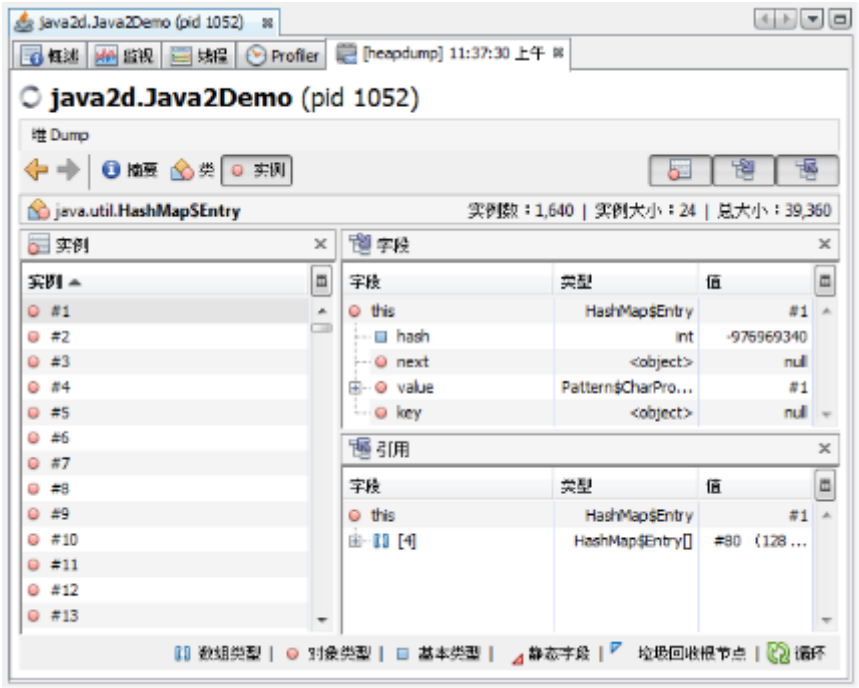
上面这些调优工具都提供了强大的功能，但是总的来说一般分为以下几类功能

堆信息查看

> 可查看堆空间大小分配（年轻代、年老代、持久代分配）
>
> 提供即时的垃圾回收功能
>
> 垃圾监控（长时间监控回收情况）



> 查看堆内类、对象信息查看：数量、类型等



> 对象引用情况查看

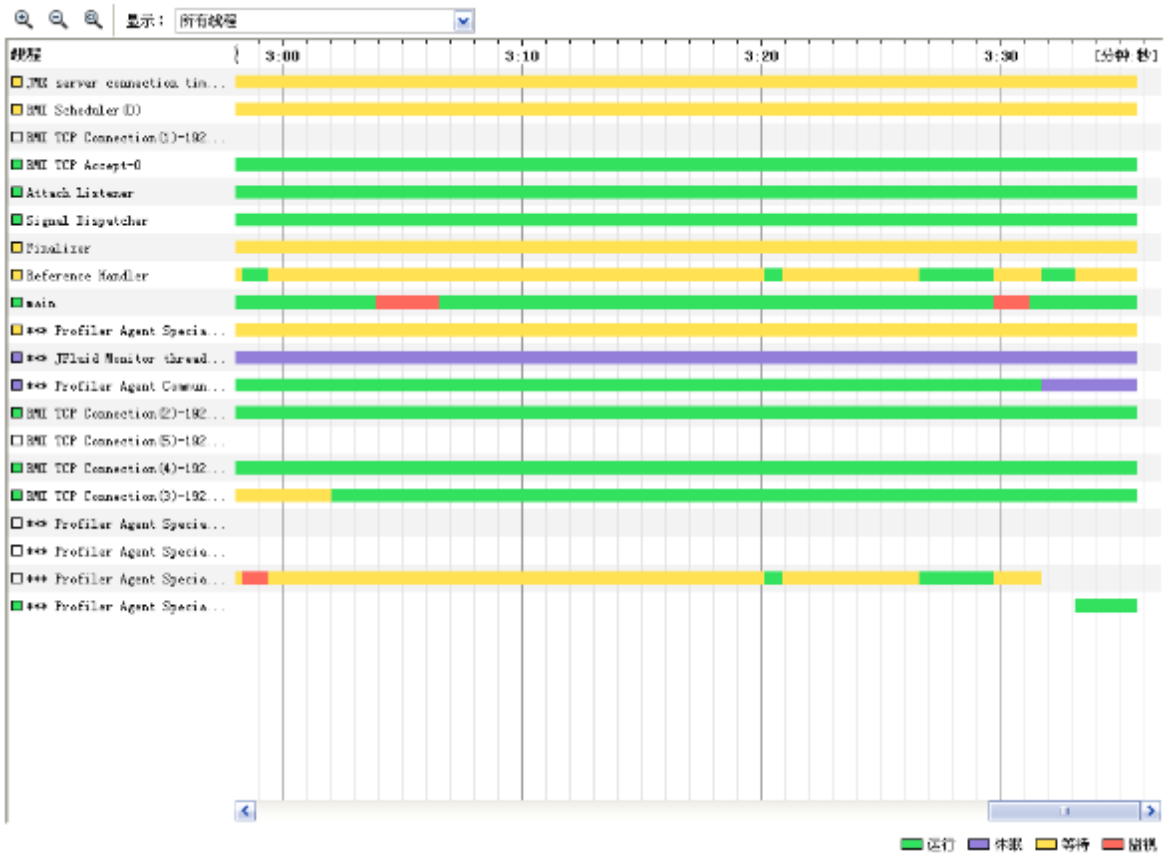
有了堆信息查看方面的功能，我们一般可以顺利解决以下问题：

-年老代年轻代大小划分是否合理

-内存泄漏

-垃圾回收算法设置是否合理

线程监控



> 线程信息监控：系统线程数量。
>
> 线程状态监控：各个线程都处在什么样的状态下

```
"Finalizer" daemon prio=8 tid=0x02ad0400 nid=0xd18 in Object.wait() [0x02c9f000..0x02c9fc94]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x22ed2b78> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(Unknown Source)
        - locked <0x22ed2b78> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(Unknown Source)
        at java.lang.ref.Finalizer$FinalizerThread.run(Unknown Source)

   Locked ownable synchronizers:
        - None

"Reference Handler" daemon prio=10 tid=0x02acf000 nid=0xb08 in Object.wait() [0x02c4f000..0x02c4fd14]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x22ed2c00> (a java.lang.ref.Reference$Lock)
        at java.lang.Object.wait(Object.java:485)
        at java.lang.ref.Reference$ReferenceHandler.run(Unknown Source)
        - locked <0x22ed2c00> (a java.lang.ref.Reference$Lock)

   Locked ownable synchronizers:
        - None

"main" prio=8 tid=0x00306800 nid=0xc00 runnable [0x0093f000..0x0093fe54]
   java.lang.Thread.State: RUNNABLE
        at CPUUserTest.longUseCpu(CPUUserTest.java:9)
        at CPUUserTest.main(CPUUserTest.java:31)

   Locked ownable synchronizers:
        - None
```

> Dump线程详细信息：查看线程内部运行情况
>
> 死锁检查

热点分析

> CPU热点：检查系统哪些方法占用的大量CPU时间
>
> 内存热点：检查哪些对象在系统中数量最大（一定时间内存活对象和销毁对象一起统计）

这两个东西对于系统优化很有帮助。

我们可以根据找到的热点，有针对性的进行系统的瓶颈查找和进行系统优化，而不是漫无目的的进行所有代码的优化。

快照分析

快照是系统运行到某一时刻的一个定格。

在我们进行调优的时候，不可能用眼睛去跟踪所有系统变化，

依赖快照功能，我们就可以进行系统两个不同运行时刻，对象（或类、线程等）的不同，以便快速找到问题

举例说，我要检查系统进行垃圾回收以后，是否还有该收回的对象被遗漏下来的了。

那么，我可以在进行垃圾回收前后，分别进行一次堆情况的快照，然后对比两次快照的对象情况。

内存泄漏检查

内存泄漏是比较常见的问题，而且解决方法也比较通用，这里可以重点说一下，而线程、热点方面的问题则是具体问题具体分析了。

内存泄漏一般可以理解为系统资源（各方面的资源，堆、栈、线程等）在错误使用的情况下，导致使用完毕的资源无法回收（或没有回收），从而导致新的资源分配请求无法完成，引起系统错误。

内存泄漏对系统危害比较大，因为他可以直接导致系统的崩溃。

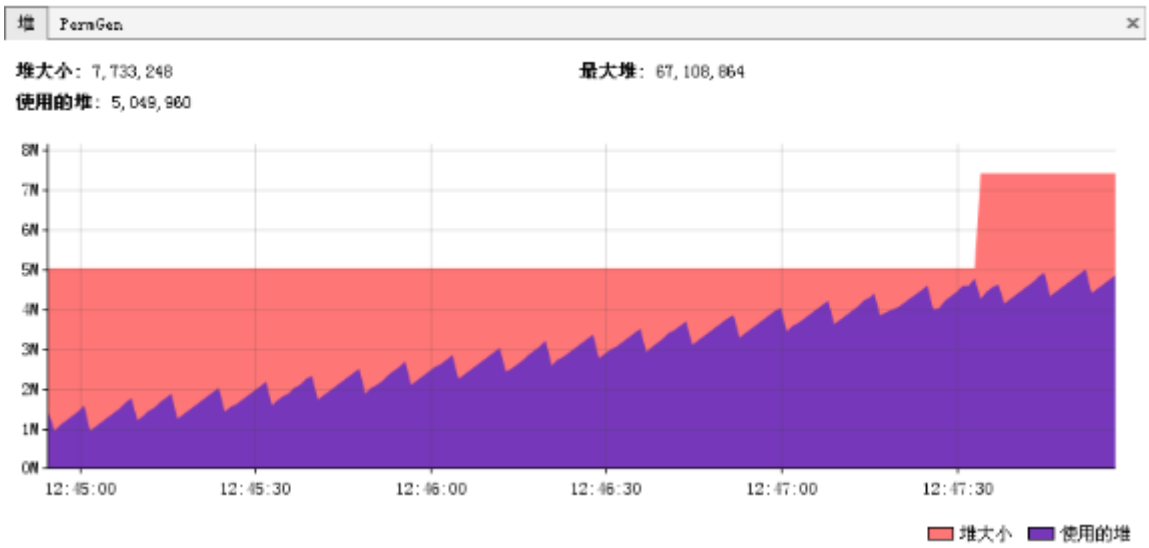
需要区别一下，内存泄漏和系统超负荷两者是有区别的，虽然可能导致的最终结果是一样的。

内存泄漏是用完的资源没有回收引起错误，而系统超负荷则是系统确实没有那么多资源可以分配了（其他的资源都在使用）。

年老代堆空间被占满

异常： java.lang.OutOfMemoryError: Java heap space

说明：



这是最典型的内存泄漏方式，简单说就是所有堆空间都被无法回收的垃圾对象占满，虚拟机无法再在分配新空间。

如上图所示，这是非常典型的内存泄漏的垃圾回收情况图。

所有峰值部分都是一次垃圾回收点，所有谷底部分表示是一次垃圾回收后剩余的内存。

连接所有谷底的点，可以发现一条由底到高的线，这说明，随时间的推移，系统的堆空间被不断占满，最终会占满整个堆空间。因此可以初步认为系统内部可能有内存泄漏。

（上面的图仅供示例，在实际情况下收集数据的时间需要更长，比如几个小时或者几天）

解决：

这种方式解决起来也比较容易，一般就是根据垃圾回收前后情况对比，同时根据对象引用情况（常见的集合对象引用）分析，基本都可以找到泄漏点。

持久代被占满

异常：java.lang.OutOfMemoryError: PermGen space

说明：

Perm空间被占满。

无法为新的class分配存储空间而引发的异常。这个异常以前是没有的，但是在Java反射大量使用的今天这个异常比较常见了。主要原因就是大量动态反射生成的类不断被加载，最终导致Perm区被占满。

更可怕的是，不同的classLoader即便使用了相同的类，但是都会对其进行加载，相当于同一个东西，如果有N个classLoader那么他将会被加载N次。因此，某些情况下，这个问题基本视为无解。当然，存在大量classLoader和大量反射类的情况其实也不多。

解决：

1. -XX:MaxPermSize=16m
2. 换用JDK。比如JRocket。

堆栈溢出

异常：java.lang.StackOverflowError

说明：这个就不多说了，一般就是递归没返回，或者循环调用造成

线程堆栈满

异常：Fatal: Stack size too small

说明：java中一个线程的空间大小是有限制的。JDK5.0以后这个值是1M。与这个线程相关的数据将会保存在其中。但是当线程空间满了以后，将会出现上面异常。

解决：增加线程栈大小。-Xss2m。但这个配置无法解决根本问题，还要看代码部分是否有造成泄漏的部分。

系统内存被占满

异常：java.lang.OutOfMemoryError: unable to create new native thread

说明：

这个异常是由于操作系统没有足够的资源来产生这个线程造成的。系统创建线程时，除了要在Java堆中分配内存外，操作系统本身也需要分配资源来创建线程。因此，当线程数量大到一定程度以后，堆中或许还有空间，但是操作系统分配不出资源来了，就出现这个异常了。

分配给Java虚拟机的内存愈多，系统剩余的资源就越少，因此，当系统内存固定时，分配给Java虚拟机的内存越多，那么，系统总共能够产生的线程也就越少，两者成反比的关系。同时，可以通过修改-Xss来减少分配给单个线程的空间，也可以增加系统总共内生产的线程数。

解决：

1. 重新设计系统减少线程数量。
2. 线程数量不能减少的情况下，通过-Xss减小单个线程大小。以便能生产更多的线程。

jvm参数优化建议

> 本质上是减少GC的次数。

如果是频繁创建对象的应用，可以适当增加新生代大小。常量较多可以增加持久代大小。对于单例较多的对象可以增加老生代大小。比如spring应用中。

GC选择，在JDK5.0以后，JVM会根据当前系统配置进行判断。一般执行-Server命令便可以。

gc包括三种策略：串行，并行，并发。

吞吐量大大应用，一般采用并行收集，开启多个线程，加快gc的是否。

响应速度高的应用，一般采用并发收集，比如应用服务器。

> 年老代建议配置为并发收集器，由于并发收集器不会压缩和整理磁盘碎片，因此建议配置：
>
> -XX:+UseConcMarkSweepGC #并发收集年老代
>
> -XX:CMSInitiatingOccupancyFraction=80 # 表示年老代空间到80%时就开始执行CMS
>
> -XX:+UseCMSCompactAtFullCollection # 打开对年老代的压缩。可能会影响性能，但是可以消除内存碎片。
>
> -XX:CMSFullGCsBeforeCompaction=10 # 由于并发收集器不对内存空间进行压缩、整理，所以运行一段时间以后会产生“碎片”，使得运行效率降低。此参数设置运行次FullGC以后对内存空间进行压缩、整理。

如何防止被Linux系统OOM (Out Of Memory Killer)暗杀

Linux内核根据服务器上当前运行应用的需要来分配内存。

因为这通常是预先发生的，所以应用并不会使用所有分配的内存。这将会导致资源浪费，Linux内核允许超分内存以提高内存使用效率。

Linux内核允许超分内存，比如总共8G内存，可以分给10个进程各1G，这通常没问题。

但问题发生在太多应用一起占用内存，有8个进程各占了1G，剩下两个进程要喝西北风了。

由于内存不足，服务器有崩溃的风险。

The server runs the risk of crashing because it ran out of memory。

为了防止服务器到达这个临近状态，内核中有一个OOM Killer杀手进程。

To prevent the server from reaching that critical state, the kernel also contains a process known as the OOM Killer。

内核利用这个杀手进程开始屠杀那些非必要进程，以便服务器正常运行。

The kernel uses this process to start killing non-essential processes so the server can remain operational.

当你认为这一切都不是问题时，因为OOM Killer只杀掉那些非必要的，不是用户需要的进程。

举例，两个应用(Apache和MySQL)通常先被杀掉，因为占用大量的内存。但这将导致一个web网站立马瘫痪了。

> **为啥某个进程被杀?**

当尝试找到为什么一个应用程序或进程被OOM killer杀掉时，有很多地方可以找到一个进程如何被杀掉以及被杀掉的原因。

1) 系统日志

$ grep -i kill /var/log/messages

host kernel: Out of Memory: Killed process 5123 (exampleprocess)

The capital K in Killed tells you that the process was killed with a -9 signal, and this typically is a good indicator that the OOM Killer is to blame.

2) 检查服务器的高低内存统计

$ free -lh

The -l switch shows high and low memory statistics, and the -h switch puts the output into gigabytes for easier human readability. You can change this to the -m switch if you prefer the output in megabytes.

同时该命令会给出Swap内存使用信息。

注意：free命令给出某个时刻得数据，需要多执行几次才能知道内存动态的占用情况。

3) vmstat可以给出某个时间段内的内存使用情况

$ vmstat -SM 10 20

20次，每次间隔10秒给出内存使用情况。

4) top命令查看内存

top 默认输出CPU的使用情况，不过你可以在top后再按下shift + M，你将得到内存的使用情况。

如何配置(Configure the OOM Killer)

1) 内存不足则重启

配置文件 /etc/sysctl.conf：

```
sysctl vm.panic_on_oom=1

sysctl kernel.panic=X
```

使用命令

```
echo “vm.panic_on_oom=1” >> /etc/sysctl.conf

echo “kernel.panic=X” >> /etc/sysctl.conf
```

大多数情况下，内存不足时每次都重启是不合适的。

2) 修改进程的优先级

既可以保护一些重要进程不被OMM killer杀掉，又可以让不重要的进程更容易杀掉:

```
echo -15 > /proc/(PID)/oom_adj (不被杀)

echo 10 > /proc/(PID)/oom_adj (更易杀)

pstree -p | grep "process" | head -1
```

3) 豁免一个进程Exempt a process

在某些情况下，豁免进程可能导致意外的行为变化，取决于系统和资源配置。

假如内核无法杀死一个占用大量内存的进程，将杀死其他进程，包括那些重要的操作系统进程。

由于OOM killer可调节的有效范围在-16到+15之间，设置为-17将豁免一个进程，因为在OOM killer调节范围之外。

通常的规则是这个参数越大越容易被杀死豁免一个进程的命令是

```
echo -17 > /proc/(PID)/oom_adj
```

4) 有风险的参数

警告：不建议用于生产环境。

假如重启，修改进程优先级，豁免一个进程不足够好，有个风险的选项：

将oom killer 功能关闭。

```
sysctl vm.overcommit_memory=2
```

使用命令

```
echo “vm.overcommit_memory=2” >> /etc/sysctl.conf
```

这一选项参数将有如下影响：

- 严重的内核恐慌kernel panic
- 系统挂住system hang-up
- 一个完整的系统崩溃system crash

为什么关闭有风险呢呢?

如果你关闭此功能，将不能避免内存耗尽。考虑此项时请极度慎重。不推荐用在生产环境

参考文献:

http://www.manongjc.com/detail/63-vghxrdhsiblevlf.html

https://www.cnblogs.com/krock/p/14421332.html

https://www.cnblogs.com/krock/p/14421332.html

https://www.cnblogs.com/pxblog/p/16102490.html

https://blog.csdn.net/lusa1314/article/details/84134458

硬核推荐：尼恩Java硬核架构班

又名疯狂创客圈社群 VIP

详情：https://www.cnblogs.com/crazymakercircle/p/9904544.html


规划中
《架构师实操篇：基于netty 手写 rpc 框架- 参考 dubbo、seata rpc框架》
《架构师实操篇：go语言学习，以及基于 go 手写 rpc 框架》
《架构师实操篇：千万级任务调度平台 架构与实操- 基于尼恩17年的亿级搜索项目》
《架构师实操篇：工业级 亿级文档搜索 平台 架构与实操- 基于尼恩17年的亿级搜索项目》
特色
会员制
提供技术方向指导，
职业生涯指导，少躺坑，少弯路
简历指导
这个很重要，
对于挪窝涨薪来说
实操性
以上项目，都是老架构师
在生产上实操过的项目
非水货
40岁老架构师，不是水货架构师
《Java高并发三部曲》为证

架构班（社群 VIP）的起源：

最初的视频，主要是给读者加餐。很多的读者，需要一些高质量的实操、理论视频，所以，我就围绕书，和底层，做了几个实操、理论视频，然后效果还不错，后面就做成迭代模式了。

架构班（社群 VIP）的功能：

提供高质量实操项目整刀真枪的架构指导、快速提升大家的:

- 开发水平
- 设计水平
- 架构水平

弥补业务中 CRUD 开发短板，帮助大家尽早脱离具备 3 高能力，掌握：

- 高性能
- 高并发
- 高可用

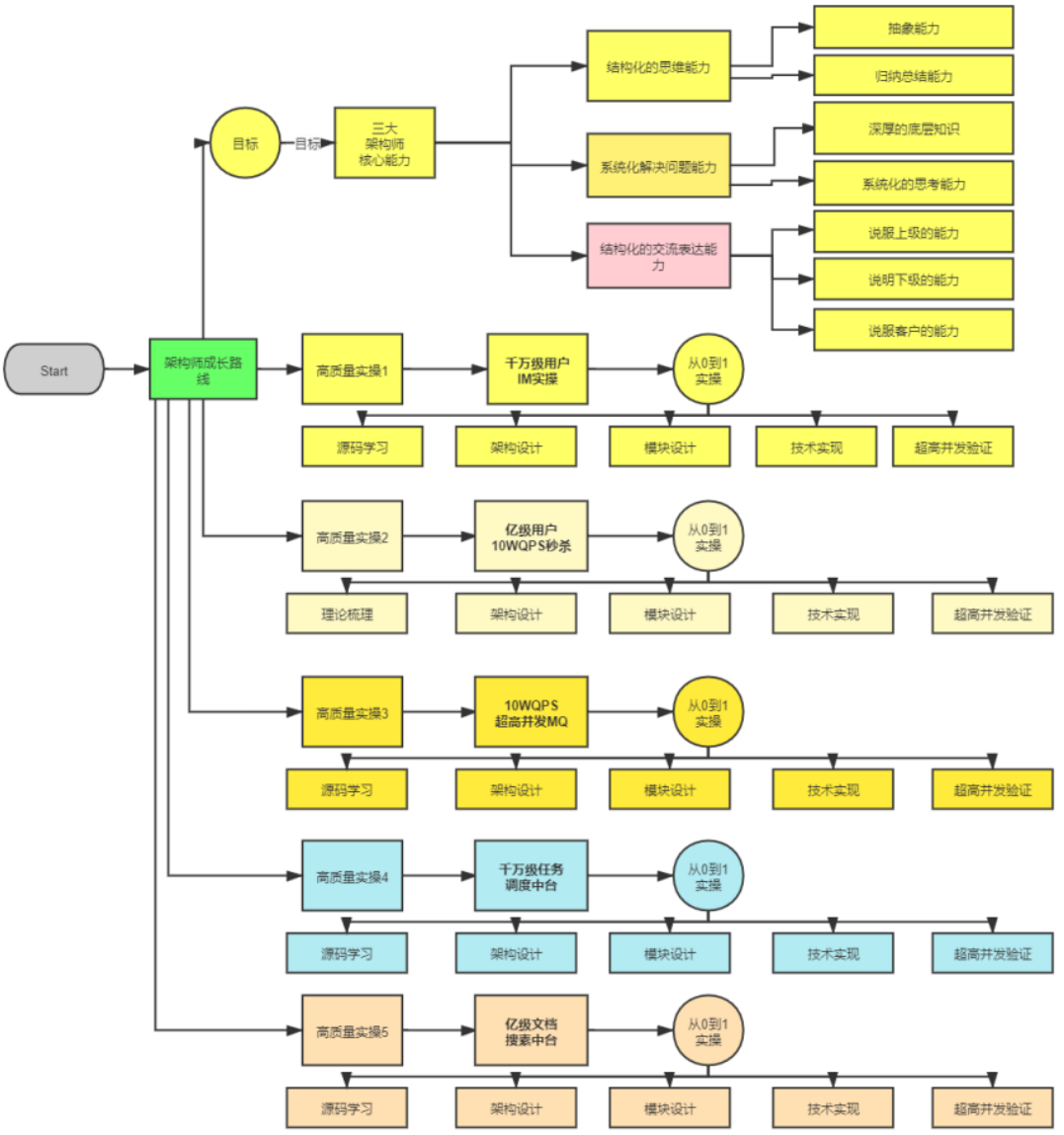
作为一个高质量的架构师成长、人脉社群，把所有的卷王聚焦起来，一起卷：

- 卷高并发实操
- 卷底层原理
- 卷架构理论、架构哲学
- 最终成为顶级架构师，实现人生理想，走向人生巅峰

架构班（社群 VIP）的目的：

- 高质量的实操，大大提升简历的含金量，吸引力，增强面试的召唤率
- 为大家提供九阳真经、葵花宝典，快速提升水平
- 进大厂、拿高薪
- 一路陪伴，提供助学视频和指导，辅导大家成为架构师
- 自学为主，和其他卷王一起，卷高并发实操，卷底层原理、卷大厂面试题，争取狠卷 3 月成高手，狠卷 3 年成为顶级架构师

N 个超高并发实操项目：简历压轴、个顶个精彩



【样章】第 17 章:横扫全网 Rocketmq 视频第 2 部曲: 工业级 rocketmq 高可用(HA)底层原理和实操

工业级 rocketmq 高可用底层原理，包含：消息消费、同步消息、异步消息、单向消息等不同消息的底层原理和源码实现；消息队列非常底层的主从复制、高可用、同步刷盘、异步刷盘等底层原理。

工业级 rocketmq 高可用底层原理和搭建实操，包含：高可用集群的搭建。

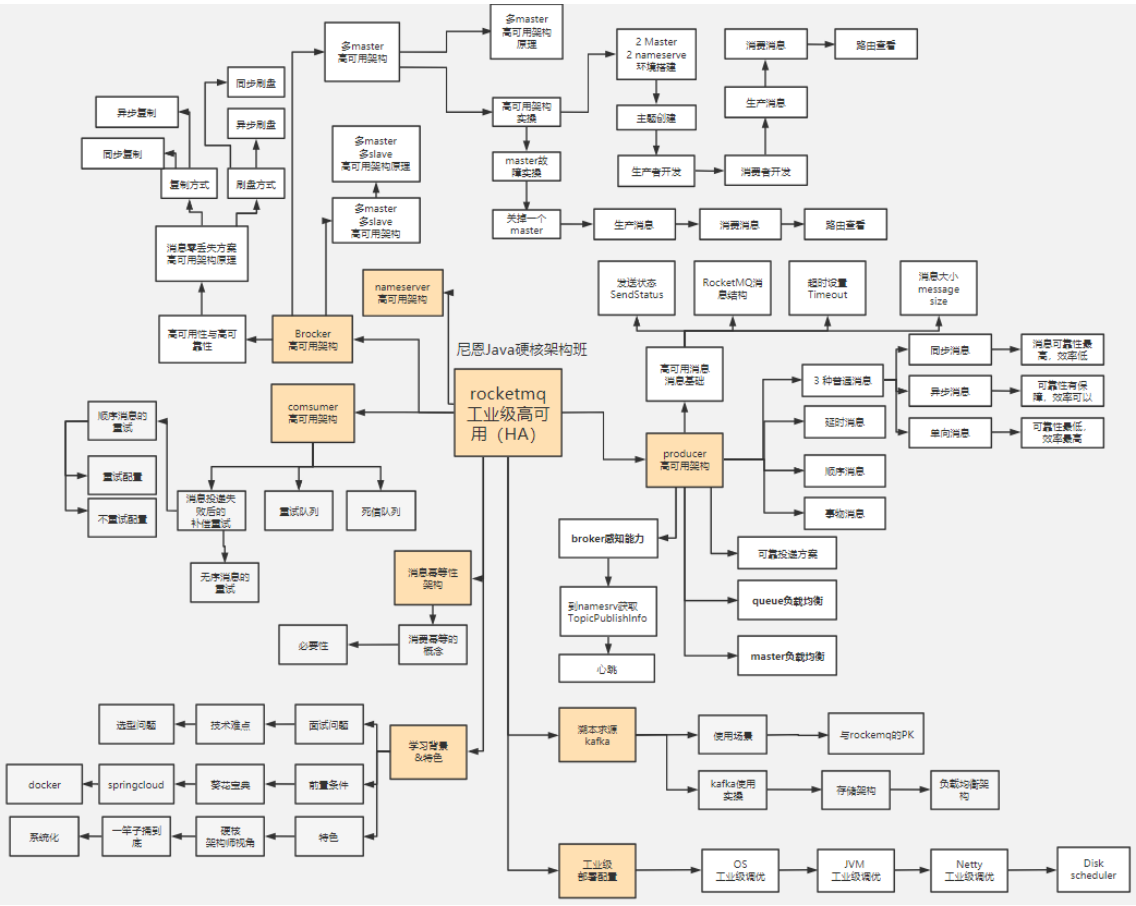
解决以下难题：

1、技术难题：RocketMQ 如何最大限度的保证消息不丢失的呢？RocketMQ 消息如何做到高可靠投递？

2、技术难题：基于消息的分布式事务，核心原理不理解

3、选型难题： kafka or rocketmq ，该娶谁？

下图链接：https://www.processon.com/view/6178e8ae0e3e7416bde9da19



成功案例：2 年翻 3 倍，35 岁卷王成功转型为架构师

详情：http://topcoder.cloud/forum.php?mod=forumdisplay&fid=43&page=1

最新　最后发表　热门　精华

成功案例：[1057号卷王] 3年小伙拿到外企offer，薪酬涨了200%
卷王1号　超级版主　前天 17:41

成功案例：[645号卷王] 4年经验卷王逆袭，被毕业后，反涨24W
卷王1号　超级版主　2022-9-21

成功案例：[878号卷王] 小伙8年经验，年薪60W
卷王1号　超级版主　2022-8-13

年薪70W案例：通过尼恩的指导，小伙伴年薪从40W涨到70W
卷王1号　超级版主　2022-2-11

成功案例：[493号卷王] 5年小伙拿满意offer，就业寒冬季逆涨30%
卷王1号　超级版主　前天 17:43

成功案例：[250号卷王] 就业极寒时代，收offer 涨25%
卷王1号　超级版主　前天 17:38

成功案例：[612号卷王] 就业极寒时代，从外包到自研
卷王1号　超级版主　前天 17:15

成功案例：[913号卷王] 热烈祝贺6年经验卷王，年薪40W
卷王1号　超级版主　2022-9-21

成功案例：[959号卷王] 4年经验卷王，喜获百度、Boss直聘等N个优质offer，最高涨100%
卷王1号　超级版主　2022-9-21

成功案例：[529号卷王] 5年经验卷王喜收2大offer，最高涨5K
卷王1号　超级版主　2022-9-21

成功案例：[811号卷王] 热烈祝贺7年经验卷王，薪酬涨30%
卷王1号　超级版主　2022-9-21

成功案例：[287号卷王] 不惧大寒潮，卷王逆市收4 offer，涨30%，可喜可贺
卷王1号　超级版主　2022-5-30

成功案例：[1002号卷王] 5月份"被毕业"，改简历后，斩获顶级央企Offer，涨薪7000+
卷王1号　超级版主　2022-7-5

成功案例：[7号卷王] 热烈祝贺小伙伴涨薪120%
卷王1号 超级版主 2022-8-13

成功案例：[134号卷王] 大三小伙卷1年，斩获顶级央企Offer，成功逆袭
卷王1号 超级版主 2022-7-6

成功案例：[1008号卷王] 5年经验卷王收42W offer，月涨8000，可喜可贺
卷王1号 超级版主 2022-5-30

成功案例：[453号卷王] 非全日制 6年卷王喜提3 offer，年薪30W，可喜可贺
卷王1号 超级版主 2022-5-21

成功案例：[924号卷王] 6年卷王喜提4 offer，最高涨薪9000，可喜可贺
卷王1号 超级版主 2022-5-21

成功案例：[15号卷王] 4年卷王入职 微软，涨薪50%，可喜可贺
卷王1号 超级版主 2022-5-12

成功案例：[527号卷王] 4年卷王喜提2 offer，涨薪50%，可喜可贺
卷王1号 超级版主 2022-5-13

成功案例：[788号卷王] 3年卷王喜提优质Offer，涨薪60%
卷王1号 超级版主 2022-5-11

成功案例：热烈祝贺：非全日制卷王，喜提2个心仪offer，面3家过2家
卷王1号 超级版主 2022-4-21

成功案例：[693号卷王] 二线城市6年卷王喜提4大优质Offer，含央企offer，最高薪酬35W
卷王1号 超级版主 2022-4-16

成功案例：[85号卷王] 双非2本小伙，春招大捷，喜提9个offer，最高薪酬近30万
卷王1号 超级版主 2022-4-14

成功案例：[741号卷王] 卷王逆袭！6年小伙从很少面试机会到搞定35K*14薪Offer
卷王1号 超级版主 2022-4-12

成功案例：[642号卷王] 热烈祝贺，6年卷王喜提优质国企offer
卷王1号 超级版主 2022-4-7

成功案例：[796号卷王] 热烈祝贺，36岁卷王喜提52万优质offer
卷王1号 超级版主 2022-3-25

成功案例：[15号卷王] 小伙卷1年，涨薪9K+，喜收ebay等多个优质offer

卷王1号 超级版主 2022-3-24

成功案例：[821号卷王] 小伙狠卷3个月，喜提10多个offer

卷王1号 超级版主 2022-3-21

成功案例：[736号卷王] 3年半经验收22k offer，但是小伙志存高远，冲击25k+

卷王1号 超级版主 2022-3-20

成功案例：热烈祝贺1群小卷王offer拿到手软，甚至拒了阿里offer

卷王1号 超级版主 2022-3-16

简历案例：简历一改，腾讯的邀请就来了！热烈祝贺，小伙收到一大堆面试邀请

卷王1号 超级版主 2022-3-10

成功案例：祝贺我圈两大超级卷王，一个过了阿里HR面，一个过了阿里2面

卷王1号 超级版主 2022-3-10

成功案例：小伙伴php转Java，卷1.5年Java，涨薪50%，喜收多个优质offer

卷王1号 超级版主 2022-3-10

成功案例：4年小伙狠卷半年，拿到 移动、京东 两大顶级offer

尼恩 超级版主 2022-3-5

成功案例：[267号卷王] 助力3年经验卷王，拿到蜂巢的17k x 14薪的offer

卷王1号 超级版主 2022-2-27

成功案例：[143号卷王] 二本院校00后卷神，毕业没到一年跳到字节，年薪45W

卷王1号 超级版主 2022-2-27

成功案例：[494号卷王] 尼恩分布式事务助力卷王拿到 中信银行offer

卷王1号 超级版主 2022-2-27

成功案例：[76号卷王] 2线城市卷王，狠卷1.5年，喜收22K offer

卷王1号 超级版主 2022-2-27

成功案例：[429号卷王] 小伙伴在社群卷5个月，涨8k+

卷王1号 超级版主 2022-2-27

成功案例：[154号卷王] 双非学校毕业卷王，连拿 京东到家&滴滴 两个大厂Offer

卷王1号 超级版主 2022-2-27

成功案例：[232号卷王] 涨薪10K，继续卷向食物链顶端

卷王1号 超级版主 2022-2-27

成功案例：狠卷1年技术，喜收 腾讯、阿里、微软三大Offer，最高年薪56W

卷王1号 超级版主 2022-2-27

成功案例：[449号卷王] 应届毕业卷王喜收 滴滴offer, 年薪33W

卷王1号 超级版主 2022-2-27

成功案例：[551号卷王] 小伙伴学完后，成功进入大厂，并且推荐自己的朋友加VIP学习

卷王1号 超级版主 2022-2-10

成功案例：[214号卷王] 助力2年经验卷王，成功拿到17K月薪

卷王1号 超级版主 2022-2-10

成功案例：[92号卷王] 课程实操助力社群小伙伴喜收 喜马拉雅Offer

卷王1号 超级版主 2022-2-10

成功案例：社群卷王小伙伴成功过了滴滴三面 获滴滴Offer

卷王1号 超级版主 2022-2-10

[612号卷王]滴滴小伙伴，蹲点考察半年，觉得靠谱后加入 疯狂创客圈

卷王1号 超级版主 2022-2-10

成功案例：[732号卷王] 尼恩助力3年经验卷王收获 京东offer, 年薪35W

卷王1号 超级版主 2022-2-27

成功案例：[558号卷王] 2年经验卷王，喜收 网易和阿里子公司两个优质offer

卷王1号 超级版主 2022-2-27

成功案例：[569号卷王] 双非应届生卷王, 喜收字节跳动实习offer

卷王1号 超级版主 2022-2-25

成功案例：[420号卷王] 狠卷1年，卷王涨薪80%，涨薪12000元!

卷王1号 超级版主 2022-2-25

成功案例：[76号卷王] 通过尼恩1年半的指导，专科学历小伙伴从0.8K涨到22K

卷王1号 超级版主 2022-2-10

简历优化后的成功涨薪案例（VIP 含免费简历优化）


1.5年小伙搞定15K offer
就业寒冬涨100%
5月7日改简历
11月21日晒offer
秘诀:
简历指导+狠狠卷
多谢啦
你好，好多公司都觉得我的简历太简单了，我感觉是不是没法优化了
简历.pdf
397.4 KB
好的
晚上指导你改
推送中台需要加不
我还没做啊，准备八股文暂时没时间搞这些，入职了会开始搞
OK
还有其他的项目吗
一个项目，太少啦
那种练习项目也行
他那个不止两倍吧，我才是两倍，原先7k，现在15k
那也很牛
都翻倍了，都是牛人
正常就30%
慢慢来
下一步可以瞄准25k
其实我工作一年
那你就更牛逼拉
一年经验，搞定15k offer，舍你其谁


卷王逆袭成功案例
6年小伙从很少面试机会到
搞定35K*14薪
3月5日改简历
4月11日拿offer
面试邀请少
小伙很苦恼
理想很丰满
目标35K
我要努力争取35+
一个月拿到了
理想的offer
面试法宝
rocketmq四部曲

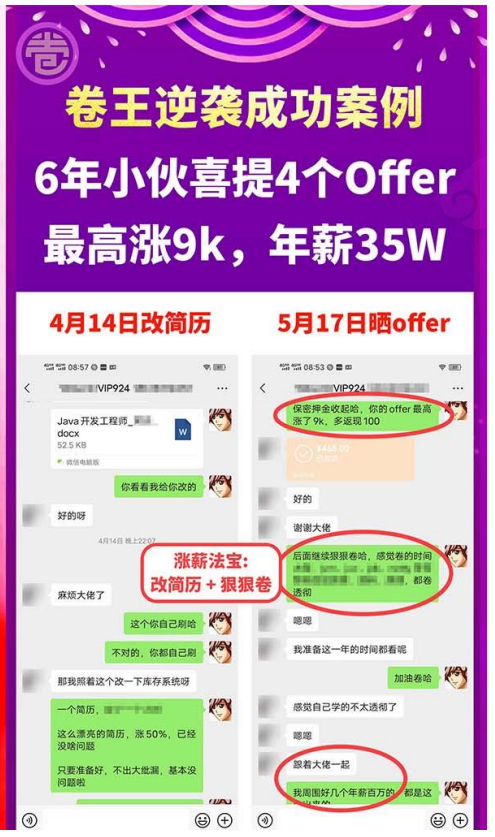

6年 经验小伙伴
喜收25K offer
3月12日改简历
12月1日晒offer
秘诀:
简历指导+狠狠卷
老师
咱们以这个项目为模板改哈
嗯好
项目有多少人，你带领多少人？
你工作几年了？
6年了
项目有多少人，你带领多少人？
语音交流一下
好
你方便吗
OK
不知道往哪个方向发展
改简历挪窝，20 涨到 25 k，高级开发，P7带的后端团队
任务重，道路远，并不是没有方向


7年经验卷王
薪酬涨30%
7月11日改简历
9月1日晒offer
秘诀:
改简历＋狠狠卷
咱们开始修改哈
要语音还是打字也行，还没到家
解决过最复杂的难题是啥？说3个哈
看看，有没有问题？
没问题
只能挪窝
下次你喊足一点哦
只要本事好，拿offer比较容易的
喊了，但是我流水太差了。。。我这家是补贴，银行不认这玩意，只能在上家的基础上加到30%封顶（特别优秀除外，我只是不错。。。）
入职了，最终并不能大幅涨薪，还是旧的太低，只涨了30%
恭喜一下
现在不比往年
能涨30%是大牛
拿到offer都算大牛
现在很多人投几百发，连个电话都没有
你已经很


4年经验卷王逆袭
被毕业后，反涨24W
7月改简历
8月30日晒offer
秘诀：
改简历＋狠狠卷

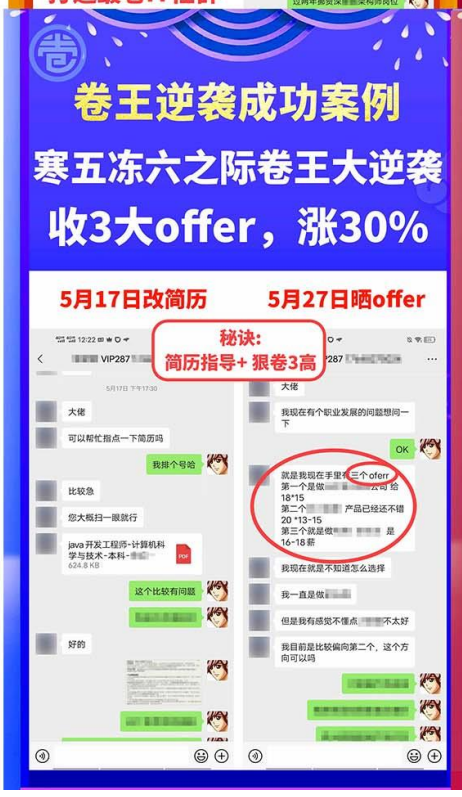

小伙5月份"被毕业"，改简历后
斩获顶级央企Offer
涨薪7000+
5月29日改简历
7月5日晒offer
秘诀：
简历指导＋狠卷3高


卷王逆袭成功案例
武汉6年喜收4个优质offer
最高的年薪35W
2月9日改简历
4月15日晒offer
面试法宝：
改简历＋实操


卷王逆袭成功案例
6年小伙喜提4个Offer
最高涨9k，年薪35W
4月14日改简历
5月17日晒offer
涨薪法宝：
改简历＋狠狠卷


卷王逆袭成功案例
5年经验小伙收2个offer
最高涨薪8k，年薪42W
5月9日改简历
5月30日晒offer
秘诀:
简历指导+ 狠卷3高
以此为样
大家狠狠卷
打造最卷IT社群


卷王逆袭成功案例
非全日制 6年经验卷王
喜提3个Offer，年包30W
5月9日改简历
5月18日晒offer
面试法宝:
改简历 + 狠狠卷


卷王逆袭成功案例
寒五冻六之际卷王大逆袭
收3大offer，涨30%
5月17日改简历
5月27日晒offer
秘诀:
简历指导+ 狠卷3高


卷王逆袭成功案例
4年卷王入职微软，涨50%
3月7日改简历
5月12日晒offer
涨薪法宝:
改简历 + 狠狠卷


4年小伙喜收百度、Boss直聘
等N个顶级Offer
最高涨幅100%
6月27日改简历
9月19日晒offer
秘诀:
改简历＋狠狠卷


卷王逆袭成功案例
4年卷王入收2个offer，涨50%
3月23日改简历
5月12日晒offer
涨薪法宝:
改简历＋狠狠卷


小伙大三暑期很焦虑
跟着尼恩卷一年
校招斩获顶级央企Offer
去年5月19日加入VIP群
今年7月5日晒offer
秘诀:
狠狠卷 书+视频


小伙高中学历
薪酬涨120%
5月6日改简历
7月22日晒offer
秘诀:
改简历＋狠狠卷


卷王逆袭成功案例
非全日制卷王 面试3家
收2个offer 涨薪30%
4月13日改简历
4月21日晒offer
面试法宝：
改简历＋面试题


5年卷王喜收2大Offer
最高涨5K
5月19日改简历
9月13日晒offer
秘诀：
改简历＋狠狠卷


卷王逆袭成功案例
3年经验卷王，涨60%
4月16日改简历
5月11日晒offer
涨薪法宝：
改简历＋狠狠卷


卷王逆袭成功案例
双非二本小伙春招大翻身
喜提9大offer
2月22日改简历
4月13日晒offer
面试法宝：
改简历＋IM实操
9大offer 最高年薪30万

修改简历找尼恩（资深简历优化专家）

- 如果面试表达不好，尼恩会提供 简历优化指导
- 如果项目没有亮点，尼恩会提供 项目亮点指导
- 如果面试表达不好，尼恩会提供 面试表达指导

作为 40 岁老架构师，尼恩长期承担技术面试官的角色：

- **从业以来，“阅历”无数，对简历有着点石成金、改头换面、脱胎换骨**的指导能力。
- **尼恩指导过刚刚就业的小白，也指导过 P8 级的老专家，都指导他们上岸。**

如何联系尼恩。尼恩微信，请参考下面的地址：

语雀：https://www.yuque.com/crazymakercircle/gkkw8s/khigna
码云：https://gitee.com/crazymaker/SimpleCrayIM/blob/master/疯狂创客圈总目录.md