JVM

请你谈谈你对jvm 的理解？，java8虚拟机和之前的变化更新

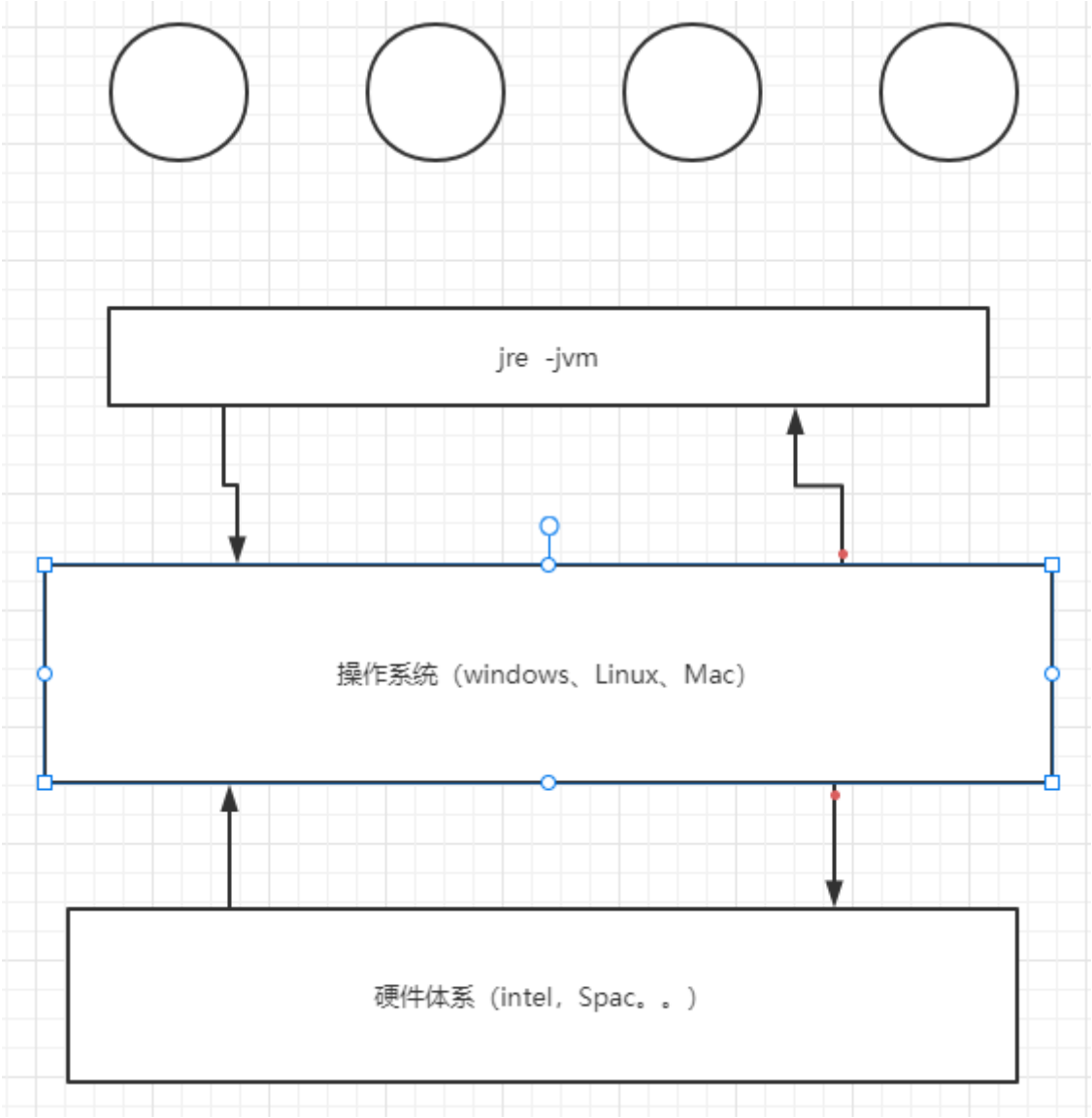
什么是OOM，什么是栈溢出StackOverFlowError？怎么分析？

JVM的常用调优

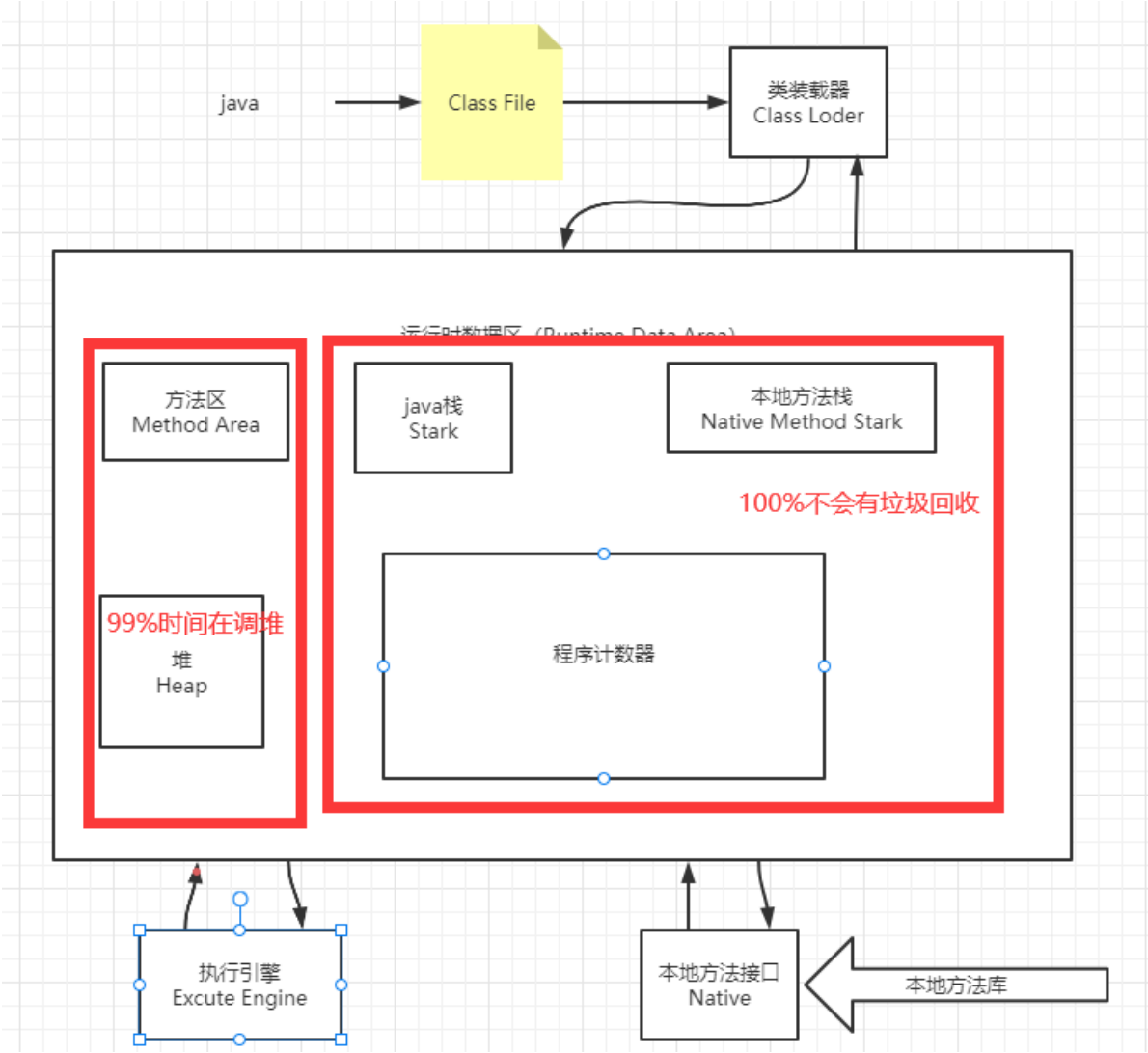
内存快照如何抓取，怎么分析DUMP文件？

谈谈JVM中，类加载器你认识吗？rt-jar ext application

1、jvm的位置

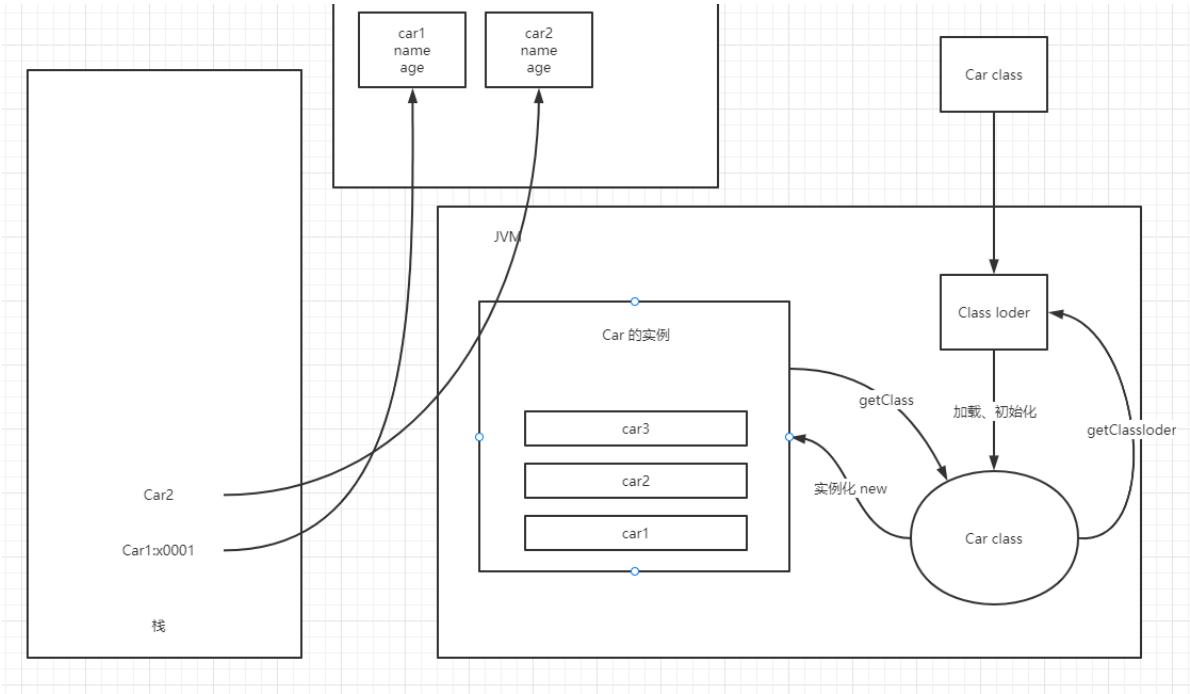


2、jvm的体系结构



3、类加载器

作用：加载Class文件 new Student();具体的实例引用在栈里，具体的人是放在堆里面的



等级：

1.虚拟机自带的加载器

2.启动类（根）加载器 BootstrapClassLoader

3.扩展类加载器 ExtClassLoader

4.应用程序（系统类）加载器 AppClassLoader

5.用户自定义加载器 CustomClassLoader

```
public class Task {
    public static void main(String[] args) {
        Task task = new Task();
        Class<? extends Task> aClass = task.getClass();

        System.out.println(aClass.getClassLoader());//AppClassLoader 应用程序加载器
        ClassLoader classLoader = aClass.getClassLoader();
        System.out.println(classLoader.getParent());//ExtClassLoader 扩展类加载器
\jre\lib\ext
        System.out.println(classLoader.getParent().getParent());//null 1、不存在
2、java程序获取不到 -rt.jar
    }
}
```

APP--》EXT--》BOOT（最终执行）

- 类加载器收到类加载的请求
- 将这个请求向上委托给父类加载器去完成，一直向上委托，直到启动类加载器
- 启动加载器检查是否能够加载当前这个类，能加载就结束，使用当前加载器，否则抛出异常，通知子类加载器进行加载
- 重复步骤

4、双亲委派机制

双亲委派机制：安全

当某个类加载器需要加载某个`.class`文件时，它首先把这个任务委托给他的上级类加载器，递归这个操作，如果上级的类加载器没有加载，自己才会去加载这个类。

5、沙箱安全机制

java安全模型的核心就是Java沙箱

什么是沙箱

沙箱是一个限制程序运行的环境，沙箱机制就是将java代码限定在虚拟机（jvm）特定的运行范围中，并且严格限制代码

沙箱组成的基本组件：

字节校验器：确保java类文件遵循java语言规则，这样可以帮助java程序实现内存保护，但不是所有的类文件都会经过字节码校验，比如类核心

类装载器：其中类装载器在3个方面对java沙箱起作用（采用的机制是双亲委派机制）

- 它防止恶意代码去干涉善意的代码
- 它守护了被信任的类库边界
- 它将代码归入保护域，确定了代码可以进行哪些操作

1、从最内层JVM自带类加载器开始加载，外层恶意同名类得不到加载从而无法使用

2、由于严格通过包来分区了访问域，外层恶意的类通过内置代码也无法获得权限访问到内层类，破坏代码就自然无法生效

- 存取控制器：存取控制器可以控制核心APP对操作系统的存取权限，而这个控制的策略设定，可以由用户指定
- 安全管理器：是核心API和操作系统之间的主要接口，实现权限控制，比存取控制器优先等级高
- 安全软件包：java.security下的类和扩展包下的类，允许用户为自己的应用增加新的安全特性，包括：
    - 安全提供者
    - 消息摘要
    - 数字签名 keytools
    - 加密
    - 鉴别

6、Native

凡是带了Native关键字的方法，说明java 的范围拿不到了，他会去调用c语言的库

会进入本地方法栈

会调用本地方法接口JNI

JNI的作用：扩展java的使用，融合不同的编程语言为java所用

他在内存区域中专门开辟了一块标记区域（登记native方法）在最终执行的时候加载本地方法库中的方法，通过JNI

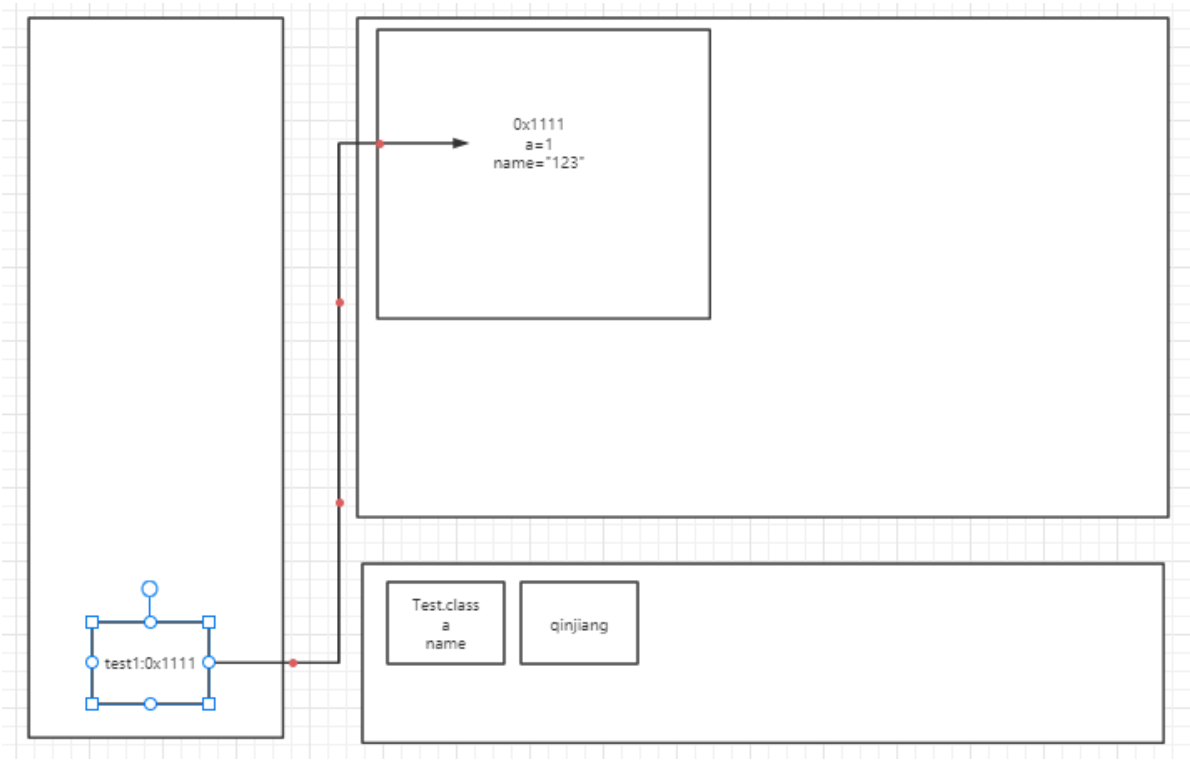
7、PC寄存器

程序计数器：program Counter Register

每个线程都有一个程序计数器，是线程私有的，就是一个指针，指向方法区中的方法字节码（用来存储指向像一条指令的地址，也即将要执行的指令代码），在执行引擎读取吓一跳指令，是一个非常小的内存空间，几乎可以忽略不计

8、方法区

方法区是被所有线程共享的，所有字段和方法字节码，以及一些特殊的方法，构造函数，接口代码也在此定义，简单说，所有定义的方法的信息都保存在该区域，此区域属于共享区间



静态变量、常量、类信息（构造方法、接口定义），运行时的常量池存在方法区中，但是实例变量存在堆内存中，和方法区无关

static、final、Class、常量池

9、栈

是一种数据结构

程序=数据结构+算法

程序=框架+业务逻辑

栈：先进后出、后进先出（桶）

队列：先进先出（管道）FIFO

喝多了吐就是栈，吃多了拉就是队列

为什么main（）方法先执行

栈：栈内存，主管程序的运行，生命周期和线程同步：线程结束栈内存也就释放，对于栈来说不存在垃圾回收问题

一旦线程结束，栈就over

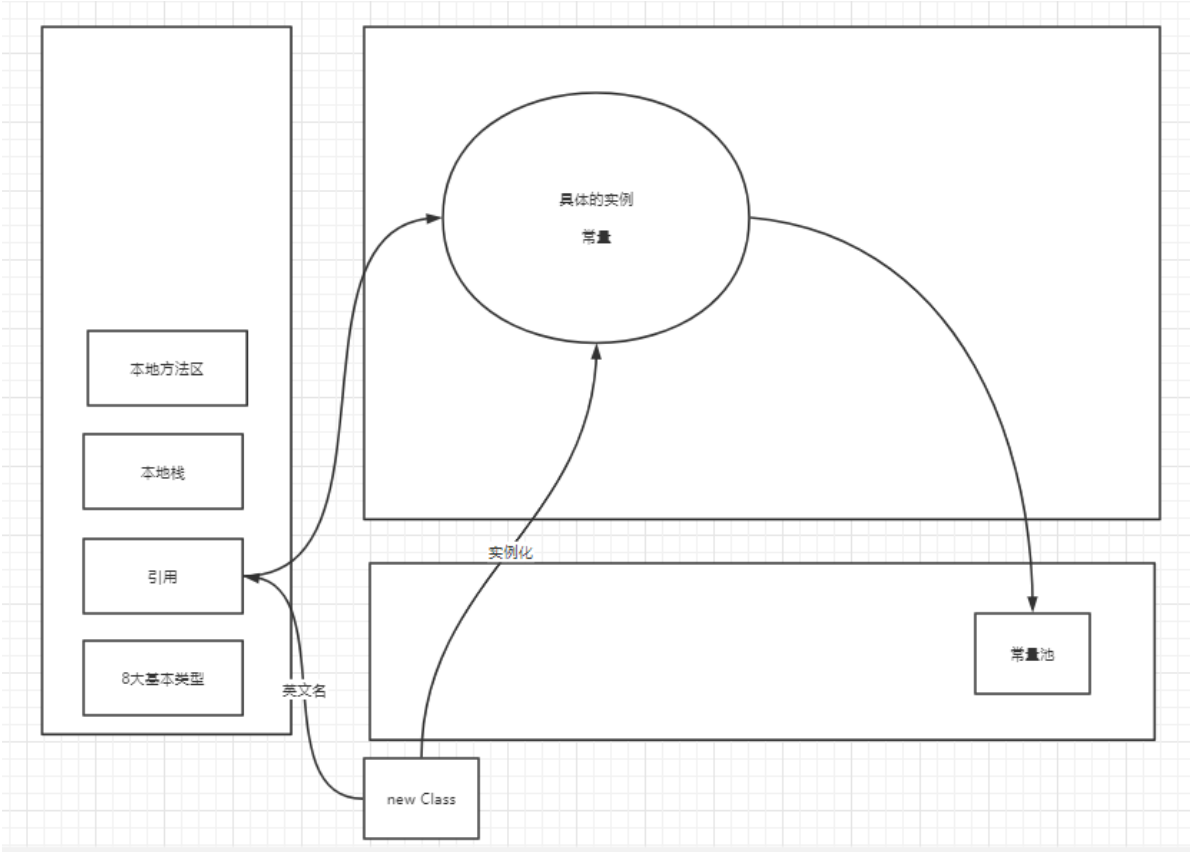
栈：八大基本类型+对象引用+实例方法

栈运行原理：栈帧

程序正在执行的方法一定在栈的顶部

栈满：StarkOverFlowError

栈+堆+方法区 的 交互关系



10、三种JVM

sun公司 HotSpot

BEA JRockit

IBM J9VM

11、堆heap

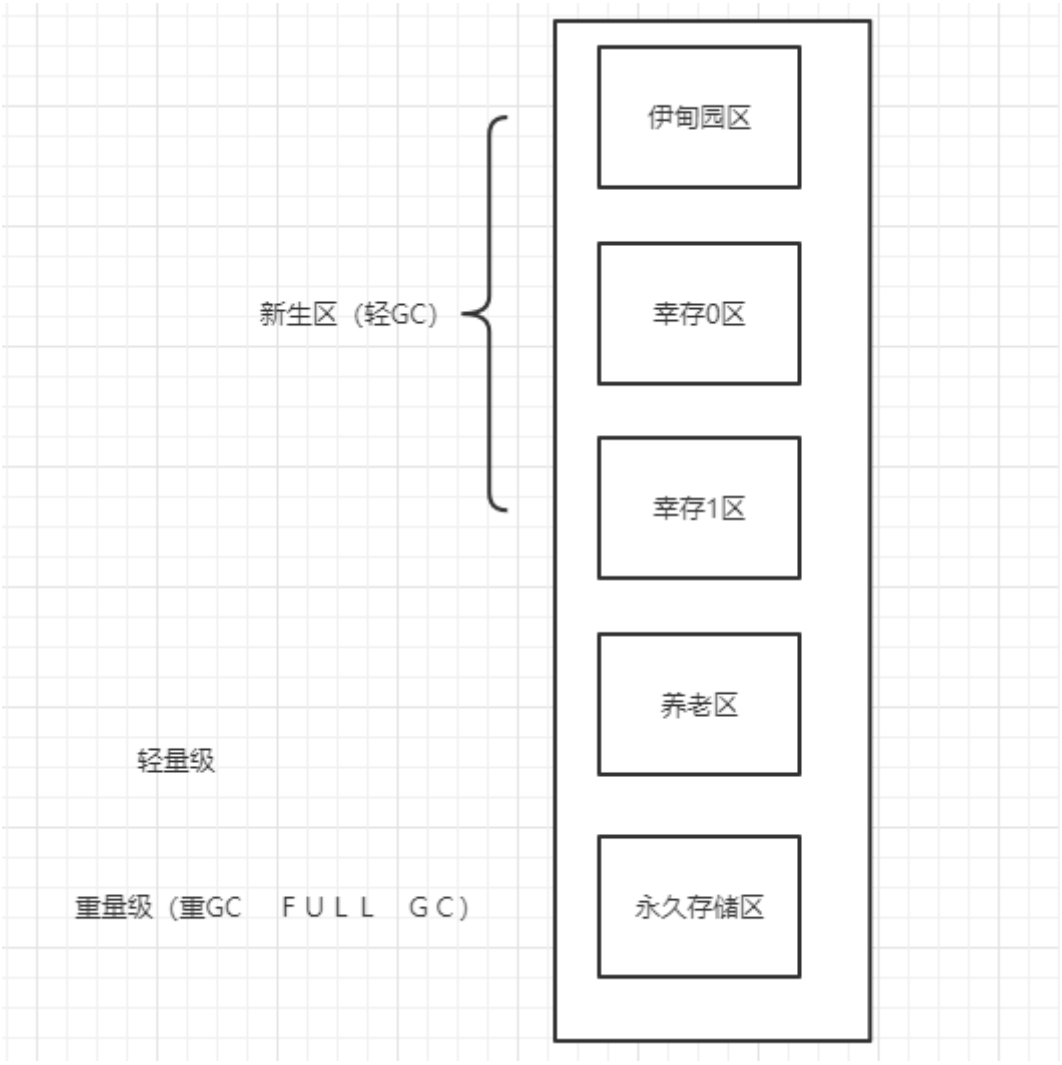
一个jvm只有一个堆内存，堆内存的大小是可以**调节**的

类加载器读取了类文件后，一般会把什么放在堆中呢

类、方法、常量、变量，保存我们所有引用类型的真实对象

堆内存中还要细分为三个区域

- 新生区（伊甸园区）
- 老年区
- 永久区



GC垃圾回收、主要是在伊甸园区和养老区

假设内存满了就会报OOM，堆内存满了

在jdk8之后永久存储区改为 元空间

12、新生区

类：诞生和成长的地方，甚至死亡

伊甸园：所有的对象都是在伊甸园区new 出来的

幸存者区（0,1）

真理：经过研究，99%的对象都是临时对象

13、老年区

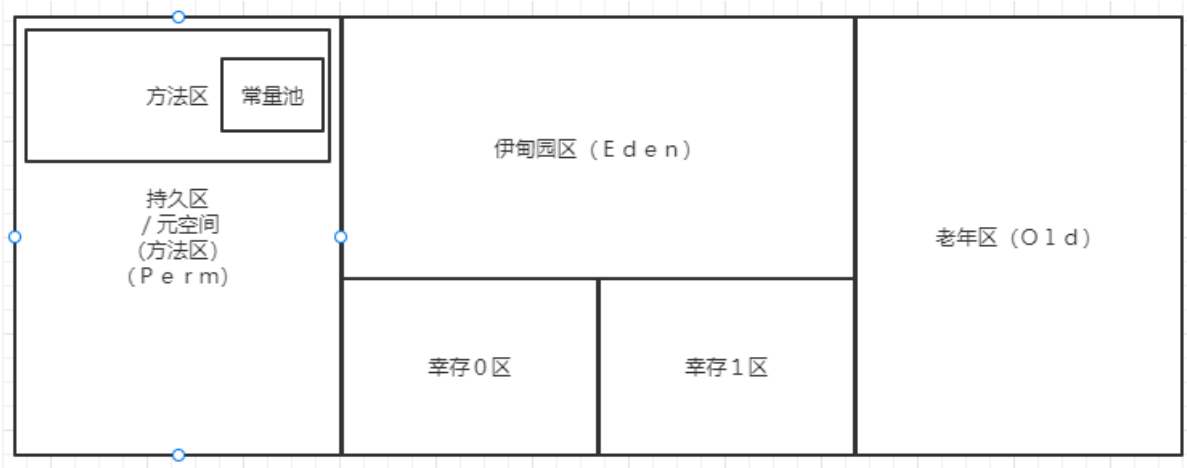
在新生区存活下来的会前往老年区

14、永久区

这个区域常驻内存，用来存放jdk自身携带的class对象，interface元数据，存储的是java运行的一些环境或类信息，这个区域不存在垃圾回收！关闭虚拟机就会释放这个区域的内存

一个启动类加载了大量的第三方jar包，Tomcat部署了太多的应用，大量动态生成的反射雷，不断地被加载，直到内存满，就会出现OOM

- jdk1.6之前：永久代，常量池在方法区中
- jdk1.7中：永久代，但是慢慢退化了，去永久代，常量池在堆中
- jdk1.8：无永久代，常量池在元空间



15、堆内存优化

-Xmx1024m -Xmx1024m -XX:+PrintGCDetails
OOM

1、尝试扩大堆内存看结果

2、分析内存，看一下哪个地方出现了问题（专业工具）

- 能够看到代码第几行出错：内存快照分析工具，MAT，jpofiler
- debug，一行行分析代码

使用JPofiler工具分析

- 分析Dump（Dump文件又叫内存转储文件或者叫内存快照文件，是进程的内存镜像，是用来给驱动程序编写人员调试驱动程序用的。dump文件中包含了程序运行的模块信息、线程信息、堆栈调用信息、异常信息等数据。）文件，快速定位内存泄露
- 获得堆中的数据
- 获得大的对象

-Xms1m -Xmx8m -XX:+HeapDumpOnOutOfMemeryError

-Xms设置初始化内存分配大小，默认1/64

-Xmx设置最大分配内存，默认1/4

16、GC 常用算法

JVM在进行垃圾回收的时候，并不是对这三个区域统一回收，大部分回收都是在新生代

- 新生代
- 幸存区（from、to）
- 老年区

GC两种：

轻GC（普通的GC）

重GC（全局GC、Full GC)

题目：

- JVM的内存模型和分区，详细到每个区放什么
- 堆里面的分区有哪些？Eden、from、to，老年区，说说他们的特点！
- GC的算法有哪些？标记清除法，标记整理，复制算法，引用计数器，怎么用的
- 轻GC和重GC分别在什么情况下发生

复制算法

幸存区--谁空谁是to

每次GC都会将Eden中存活下的对象转移到幸存区中：一旦伊甸园区被GC后就是空的

如果两个区都有对象，将其中一个幸存区中的数据复制到另一个幸存区中从而使得必然会出现一个to区的算法

当一个对象经历了15次GC之后依然存活，就会进入老年区

-XX:MaxTenuringThreshold=999

通过这个参数可以设定进入老年代的时间

好处：没有内存的碎片

坏处：浪费了内存空间：多了一半的to空间永远是空的，假设对象100%存活（极端情况）复制算法成本会很高

复制算法最佳使用场景，对象存活度较低的时候，新生区

标记清除法

扫描这些对象，对活着的对象进行标记

扫描这些对象，对没有标记的对象进行清除

缺点：两次扫描严重浪费时间，需要时间成本，会产生内存碎片

优点：不需要额外的空间

标记压缩

再优化：

压缩：防止内存碎片的产生，再次扫描，向一端移动存活的对象

标记清除压缩

进行5此标记清除之后再次进行压缩算法

总结

内存效率：复制算法>标记清除算法>标记压缩算法（时间复杂度）

内存整齐度：复制算法=标记压缩算法>标记清除算法

内存利用率：标记压缩算法=标记清除算法>复制算法

难道没有最优的算法吗

没有，只有最合适的--》GC：分代收集算法

年轻代：

- 存活率低
- 复制算法

老年代：

- 区域大，存活率高
- 标记清除（内存碎片不是太多的时候）+标记压缩混合实现

17、JMM（java memery model=java内存模型）

什么是JMM

【JMM】（Java Memory Model的缩写）

他是干嘛的

作用：缓存一致性协议，用于定义数据读写的规则

JMM定义了线程和主内存之间的抽象关系：线程之间的共享变量存储在主内存中，每个线程都有一个私有的本地内存

解决共享对象可见性问题：volilate

该如何学习

JMM抽象的概念理论

JMM对这八种指令的使用，制定了如下规则：

- 不允许read和load、store和write操作之一单独出现，即使用了read必须load，使用了store必须write
- 不允许线程丢弃它最近的assign操作，即工作变量的数据改变了之后，必须告知主存
- 不允许一个线程将没有assign 的数据从工作内存同步回主存
- 一个新的变量必须在主存中诞生，不允许工作内存直接使用一个未被初始化的变量，就是对变量实施use、store操作之前，必须经过assign和load操作
- 一个变量同一时间只有一个线程能对其进行lock，多次lock后，必须执行相同次数的unlock才能解锁
- 如果对一个变量进行lock操作，会清空所有工作内存中此变量的值，在执行引擎使用这个变量前，必须重新load或者assign操作初始化变量的值
- 如果一个变量没有被lock，就不能对其进行unlock操作，也不能unlock一个被其他线程锁住的变量
- 对一个变量进行unlock操作之前，必须把此变量同步回主内存

volilate