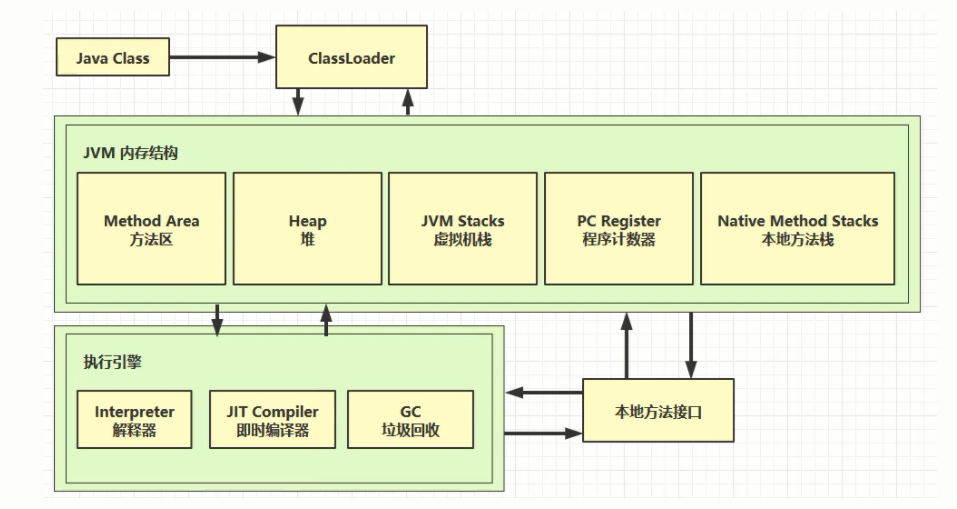
# JVM整体结构



# 内存结构

## 程序计数器

### 定义

### 作用

记住下一条JVM指令的地址。



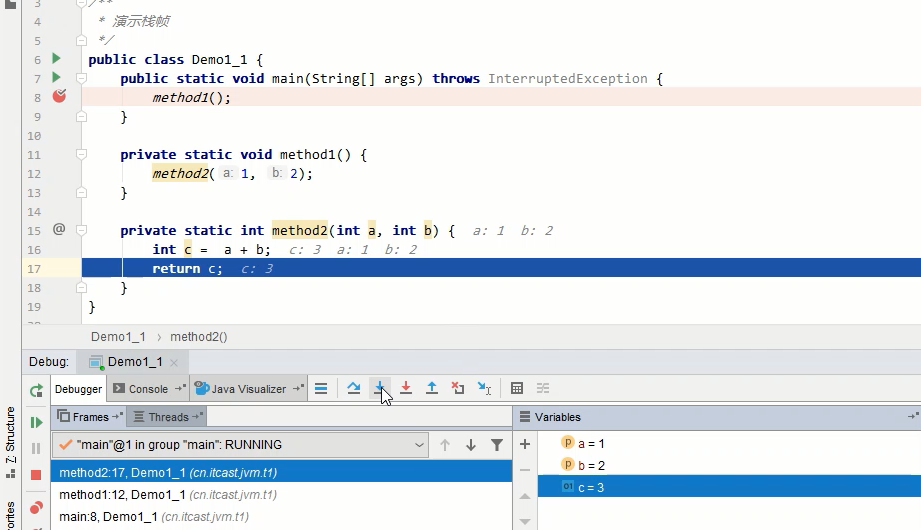
### 特点

1. 线程私有。每个线程都有自己的程序计数器；
2. 不会存在内存溢出。JVM规范中唯一一个不会出现内存溢出的区。

## 虚拟机栈

### 定义

1. java虚拟机栈，Java Virtual Machine Stack；
2. 每个线程运行时所需要的内存，成为虚拟机栈。
3. 每个栈由多个栈帧组成，对应每次方法调用时占用的内存。
4. 每个线程只能有一个活动栈帧，对应当前正在执行的那个方法。



### 问题辨析

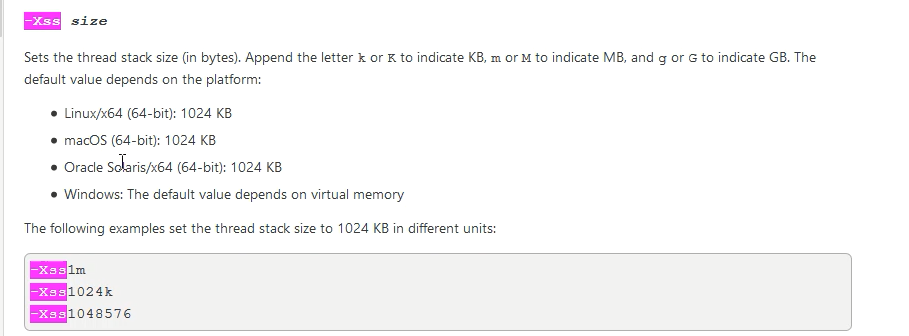
1. 垃圾回收是否涉及栈内存？

不涉及，栈内存就是一个个方法执行的栈帧内存组成。栈帧内存在每次方法调用结束后会自动弹出栈，被回收掉。

1. 栈内存分配越大越好吗，程序跑的越快呢？

不一定，栈内存是每个线程占用的内存，栈内存分配越大，意味着线程数会变小（虚拟机物理内存是固定的）。

可以通过 -Xss 参数设置栈内存大小。



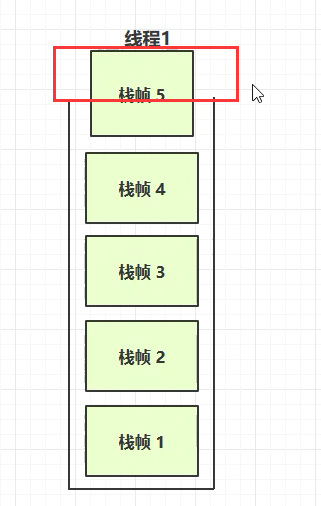
1. 方法内的局部变量是否线程安全？
   1. 如果方法内局部变量没有逃离方法的作用范围，它是线程安全的；
   2. 如果是局部变量引用了对象，并逃离了方法的作用方法，需要考虑线程安全。



### 栈内存溢出

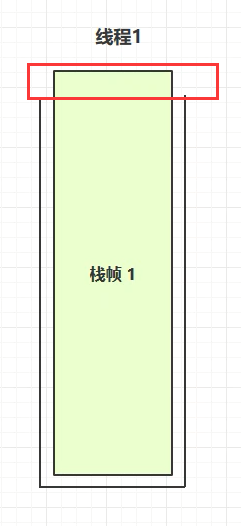
#### 什么情况会导致栈内存溢出？

1. 栈帧过多，导致栈内存溢出；



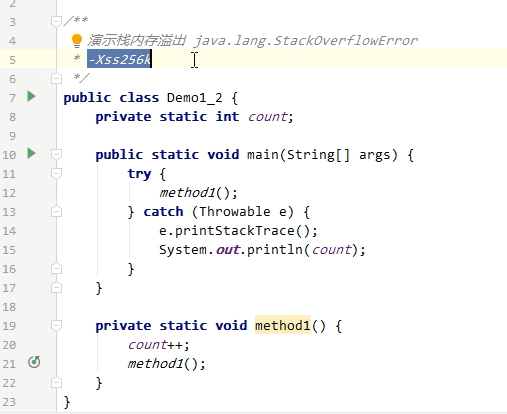
（1）方法的递归调用，没有合理的结束条件，一直加入栈帧，没有弹出；

1. 栈帧内存大于栈内存。一般不会出现



#### 常见的栈内存溢出场景

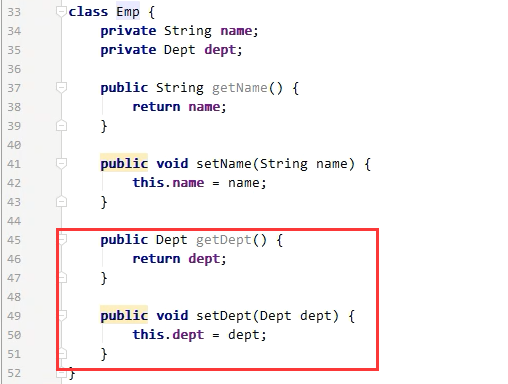
1、方法没有正确结束递归，循环调用

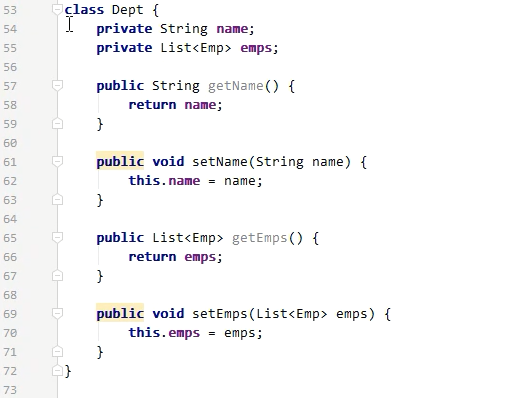


可以通过JVM -Xss 设置栈内存大小。



栈内存溢出保错：java.lang.StackOverflowError



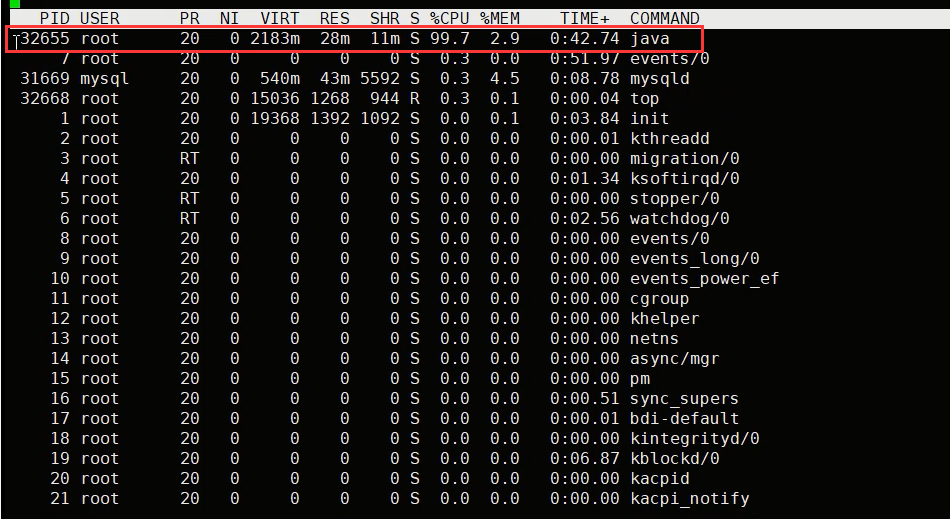




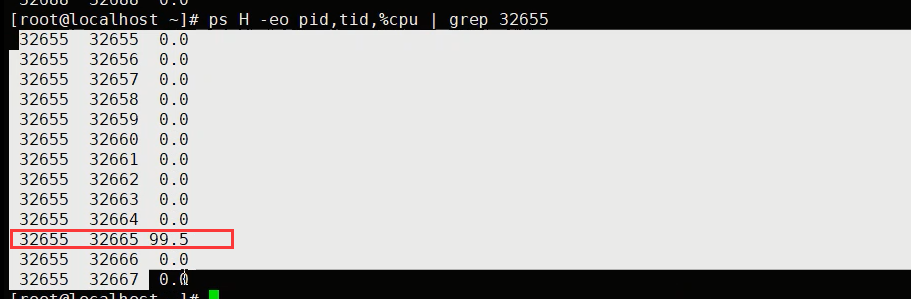
### 线程运行诊断

#### 案例1：cpu占用过高

1.使用top命令，查看是哪个进程对cpu的占用清况，找到哪个进程多cpu占用过高。

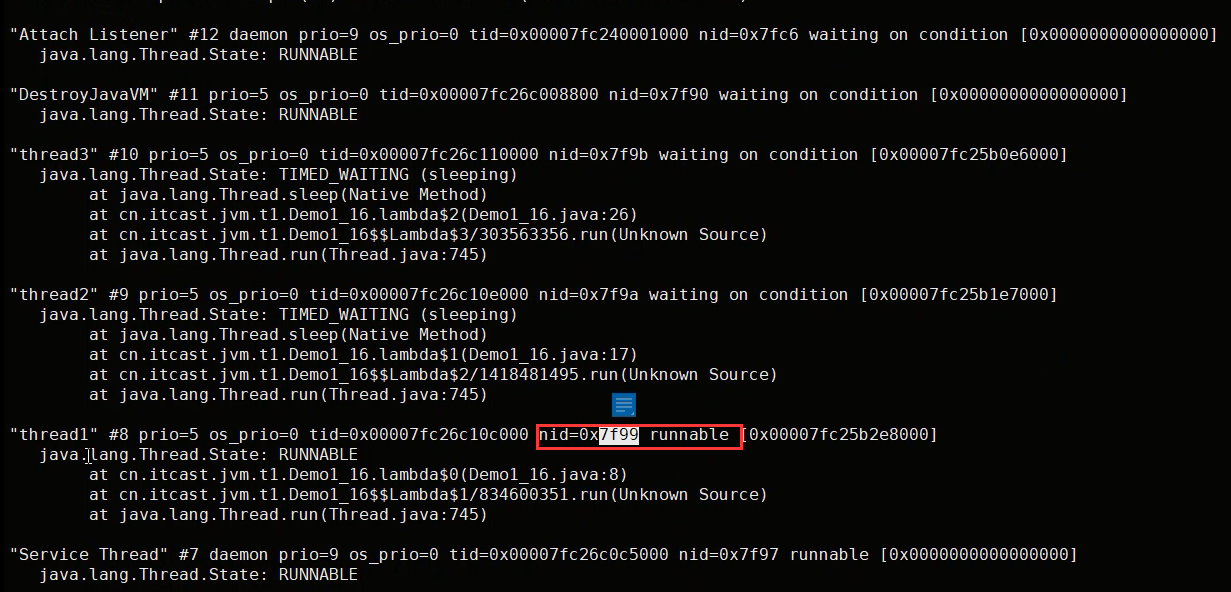


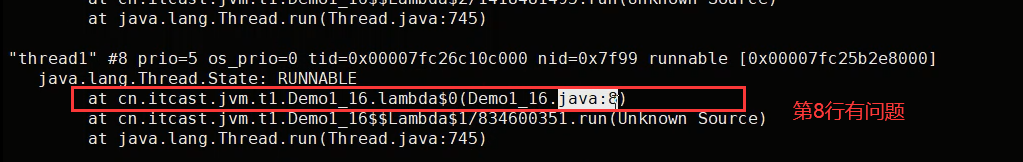
2.使用ps 命令进一步定位是哪个线程对cpu的占用过高



1. 使用jdk 提供的命令 jstack 进程ID ,查看jvm中运行的线程情况

Jstack输出的是16进制的，需要将2中的10进制线程id转为16进制。

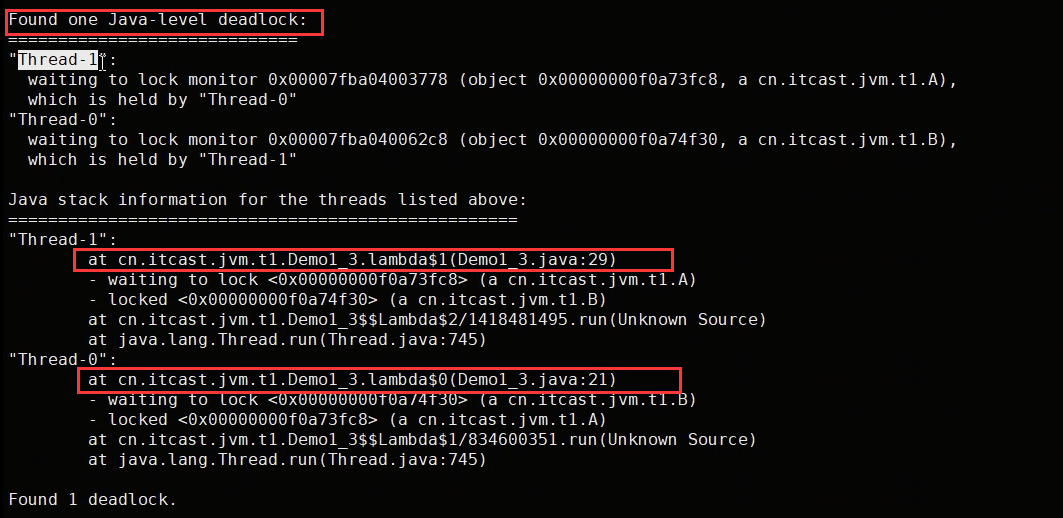




#### 案例2：程序运行很长时间没有结果

可能是由于线程死锁导致

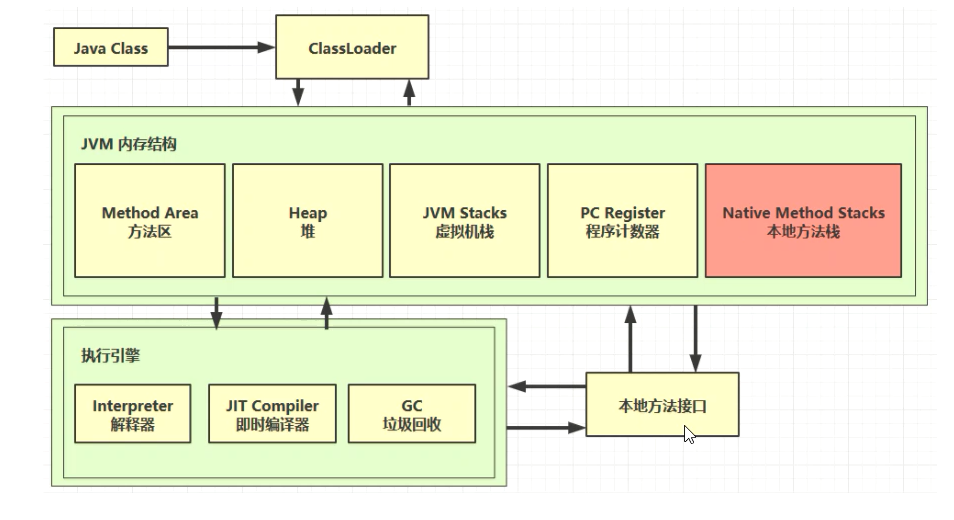
jstack 进程ID 查看进行中线程情况

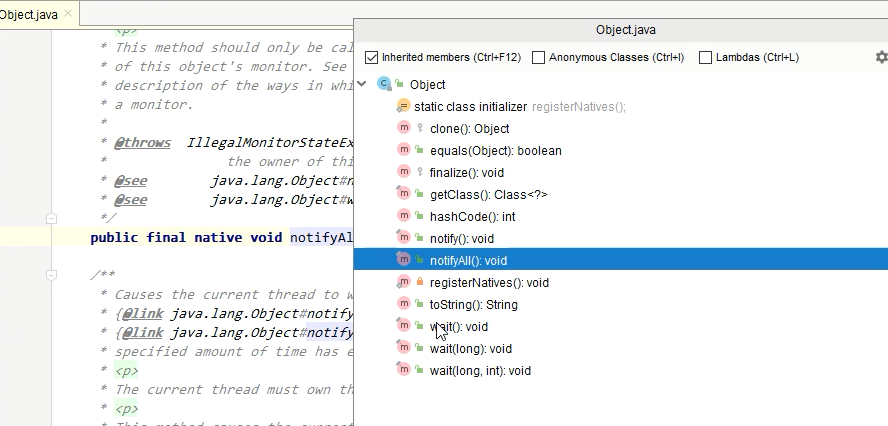




## 本地方法栈

Native Method Stack





Java 通过本地方法调用C或C++底层API与操作系统打交道。

## 堆 Heap

通过new关键字，创建出来的对象都会使用堆内存。

特点：

1. 线程共享，堆中的对象都要考虑线程安全问题。
2. 有垃圾回收机制，

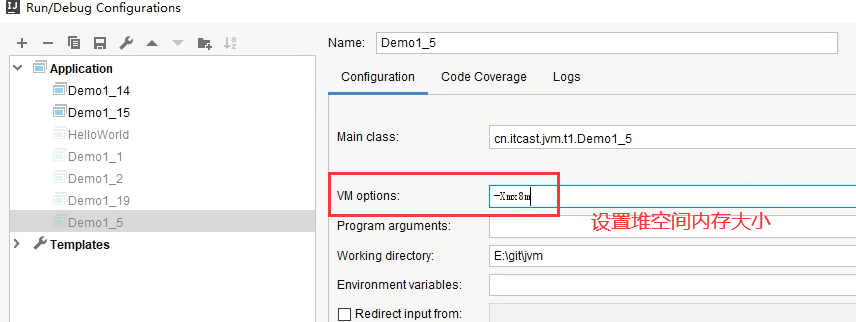
### 堆内存溢出

堆中的对象不再被使用时，会被垃圾回收。但当堆中的对象一直被占用，而且一直在产生新的对象，就会导致堆内存溢出。



堆内存溢出错误：Java.lang.OutOfMemeryError ：Java heap space

可以通过jvm -Xmx 参数设置堆内存大小。

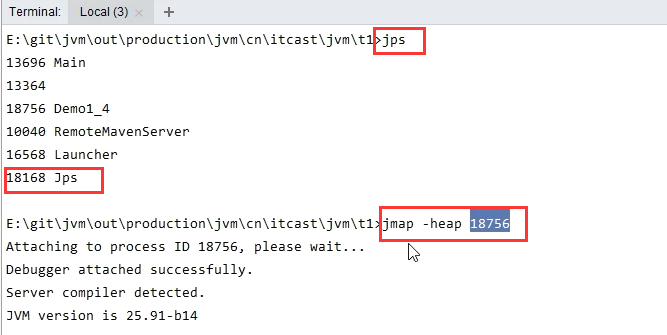


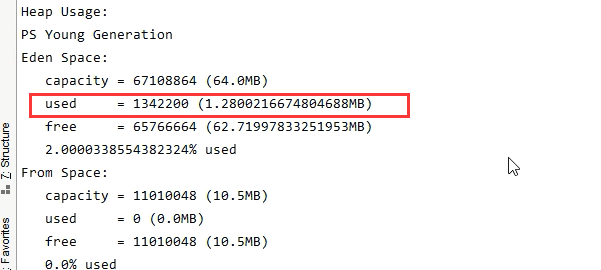
### 堆内存诊断

1. Jps 工具
   1. 查看当前系统中有哪些Java进程
2. Jmap 工具
   1. 查看堆内存占用情况
3. Jconsole 工具
   1. 图形化界面，多功能检测，可以连续检测

jmap使用

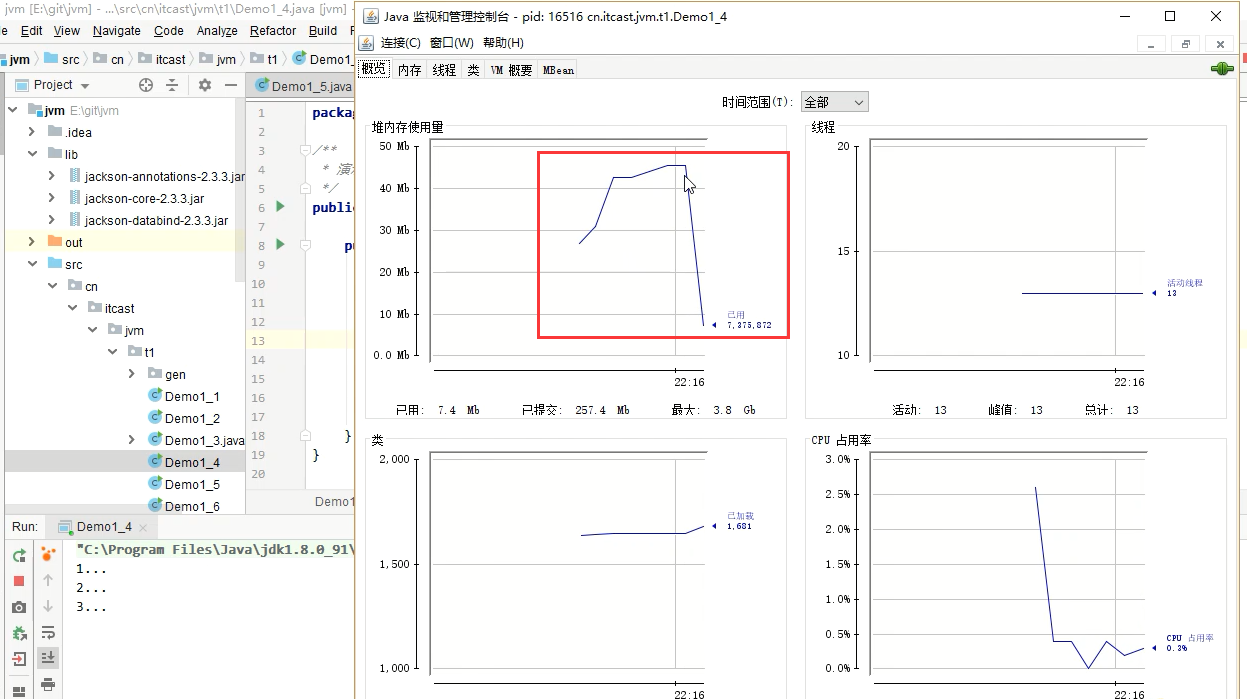






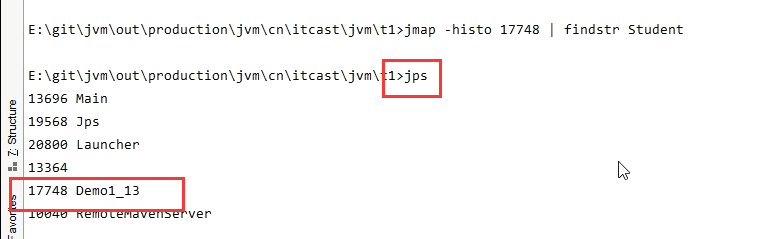
jconsole

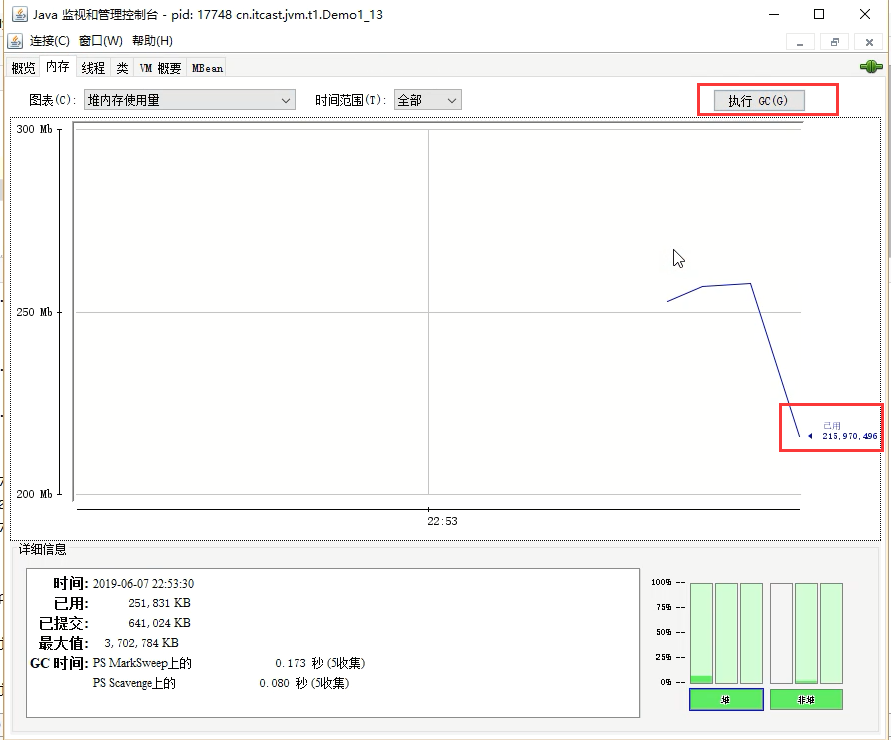


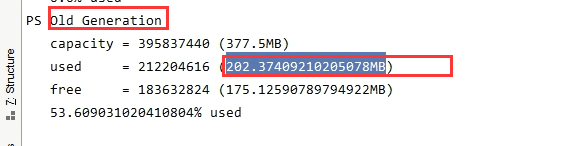


### 案例

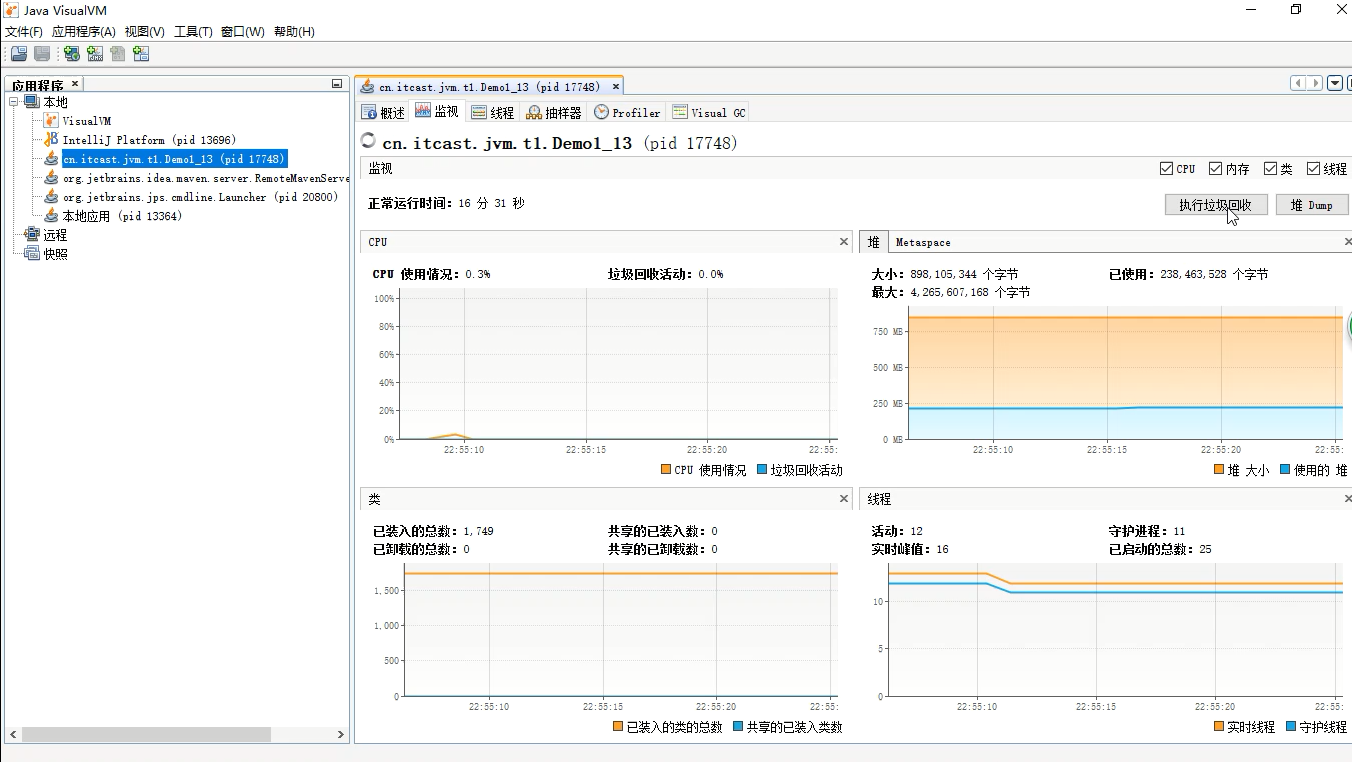
垃圾回收后，内存占用依旧很高

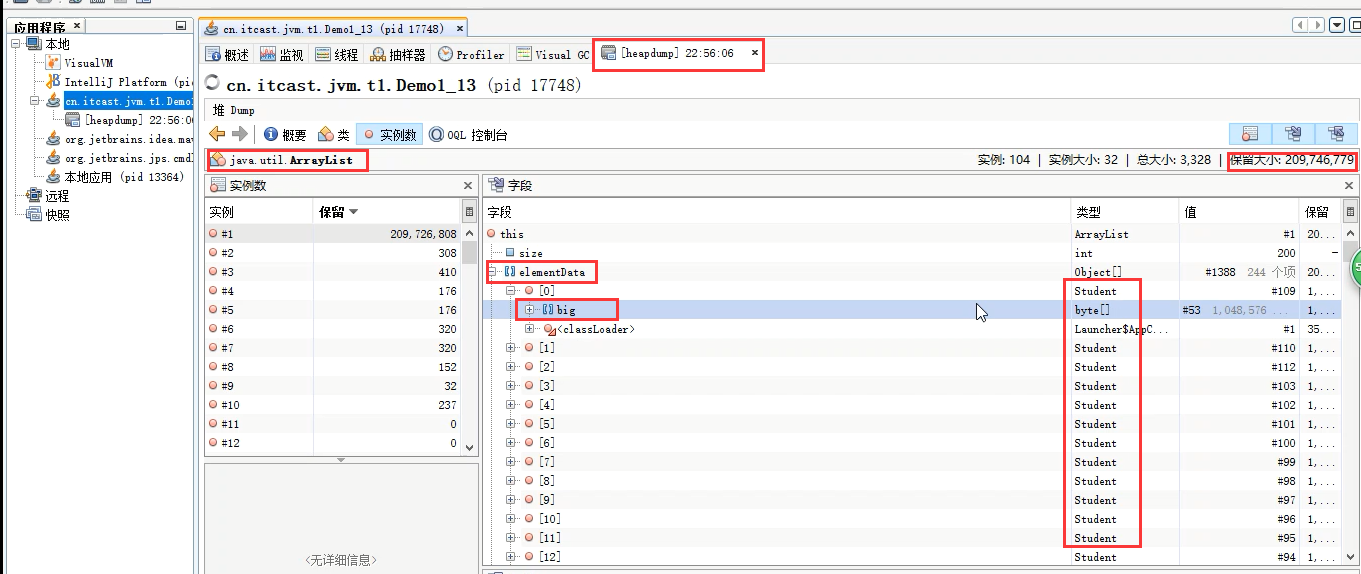


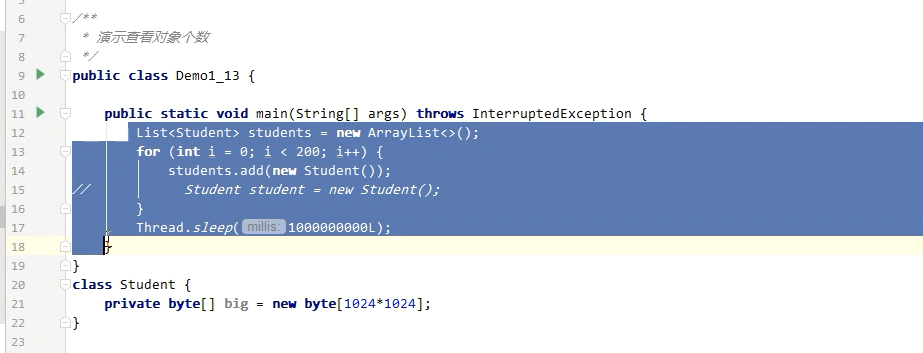




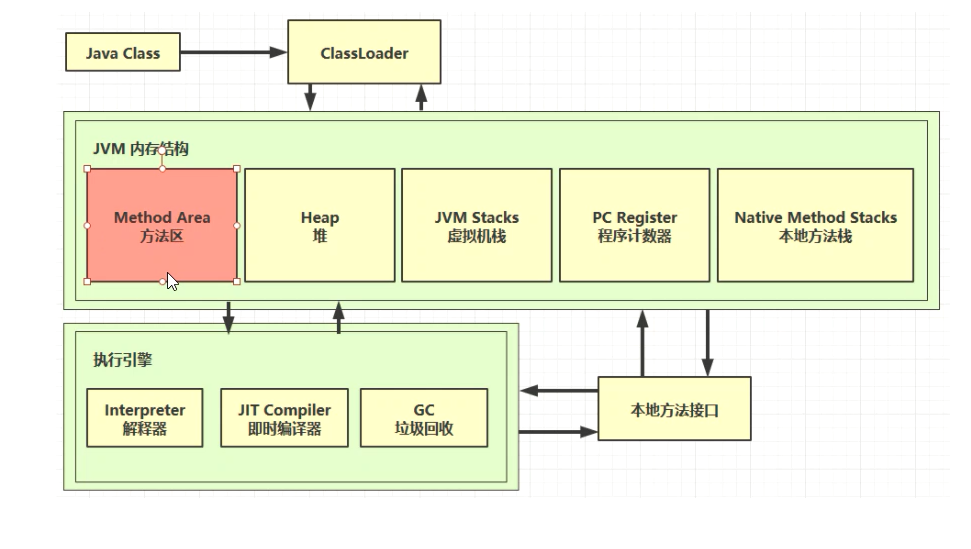
Jvisualvm

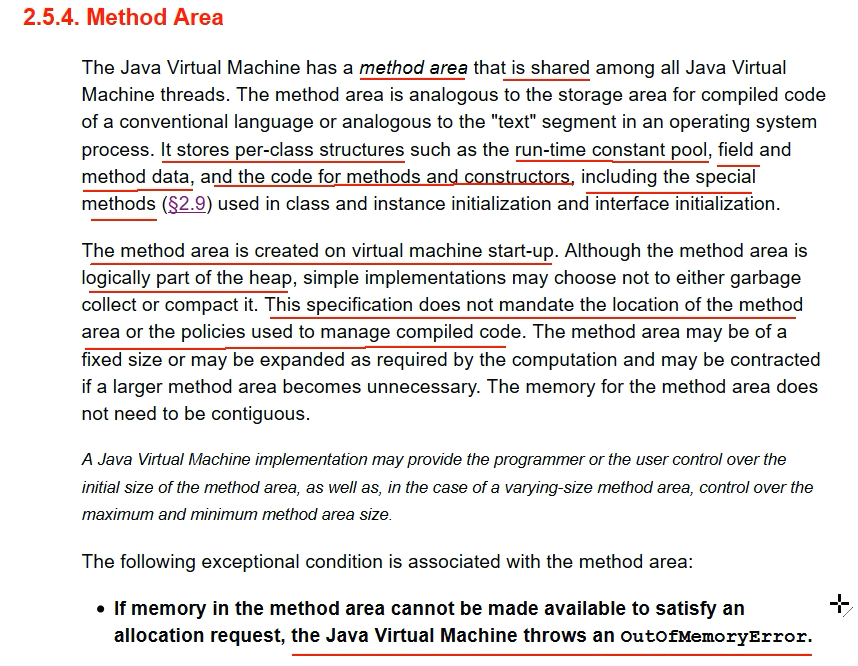




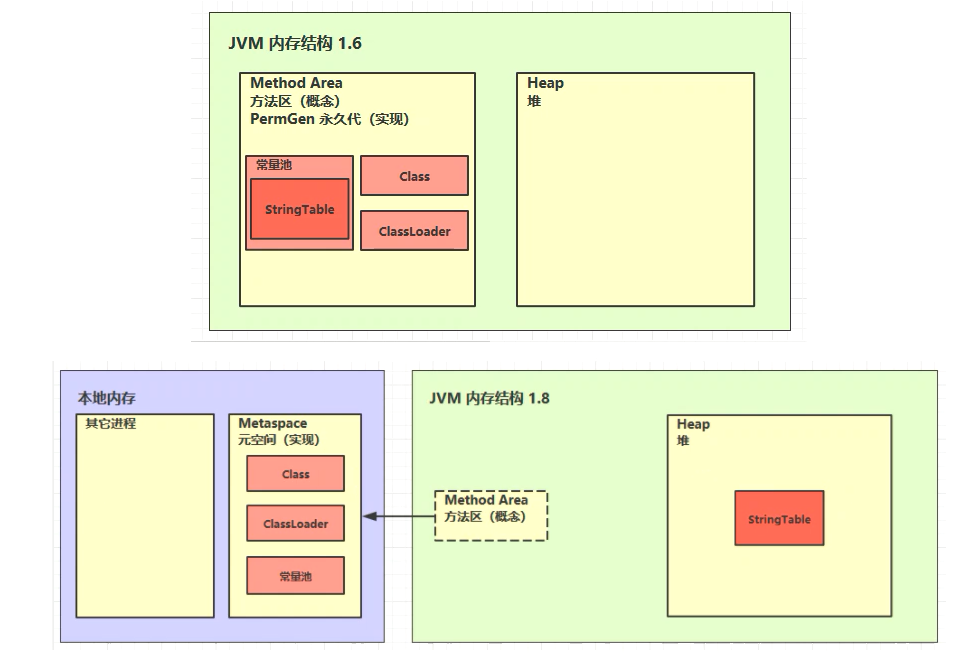


## 方法区



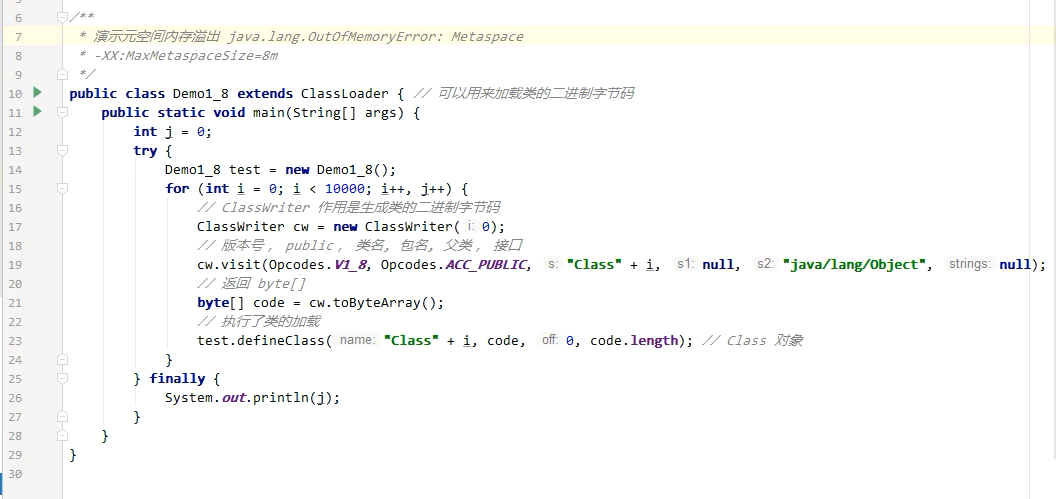


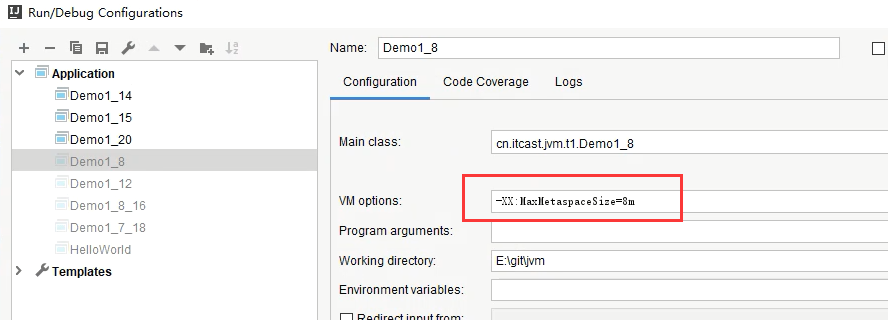
### 方法区的组成



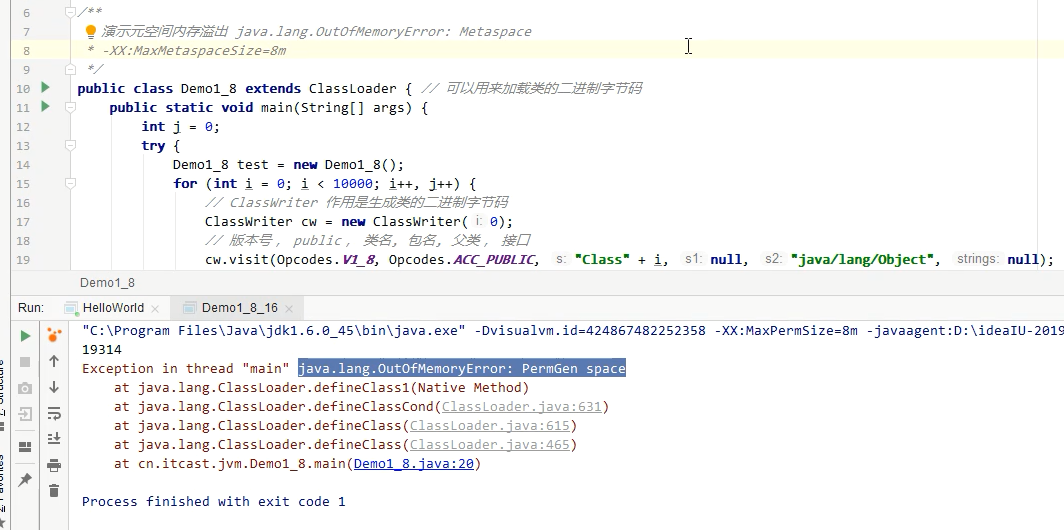


Jdk 1.8以后



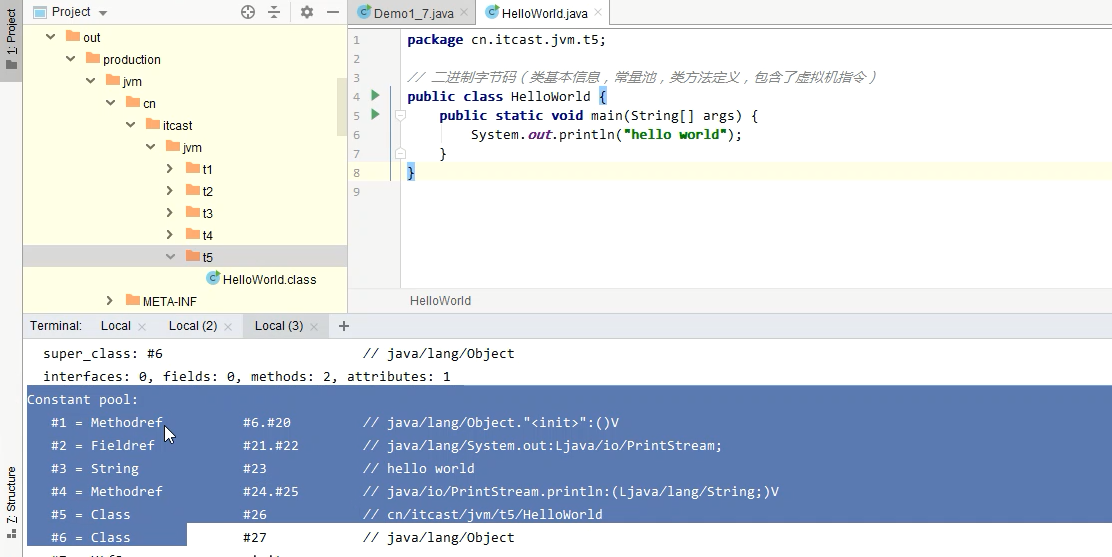


Jdk 1.8以前

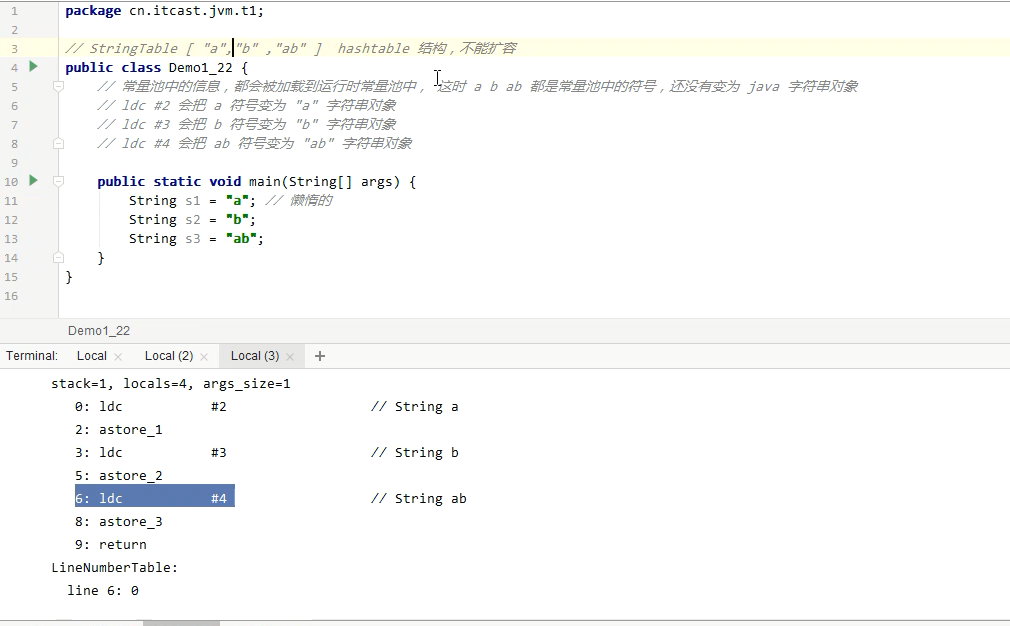


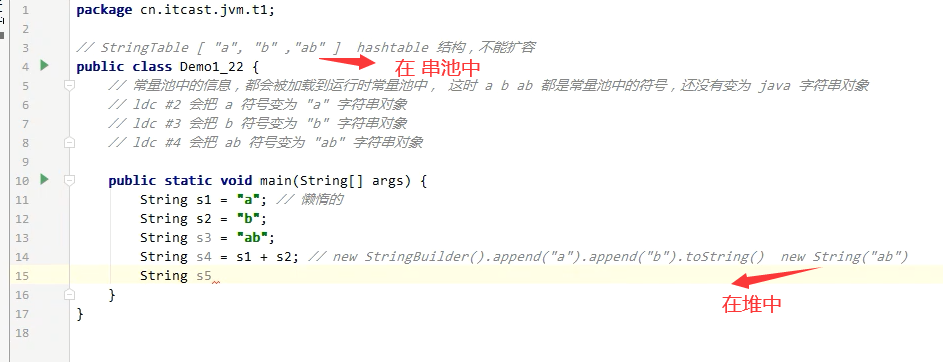
## 运行时常量池

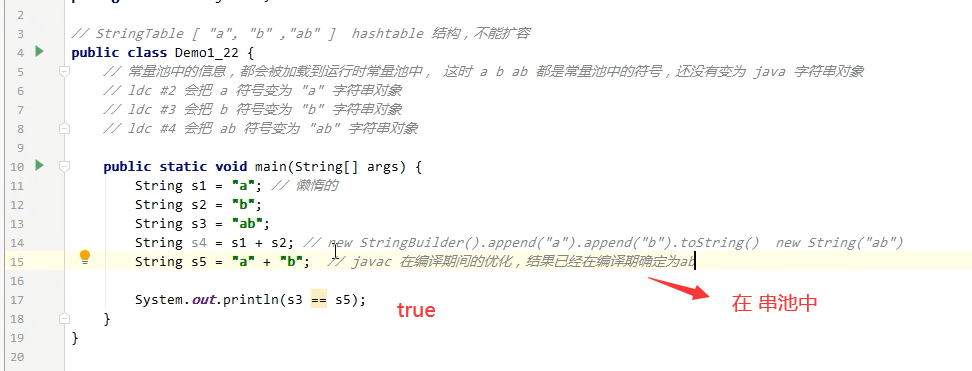


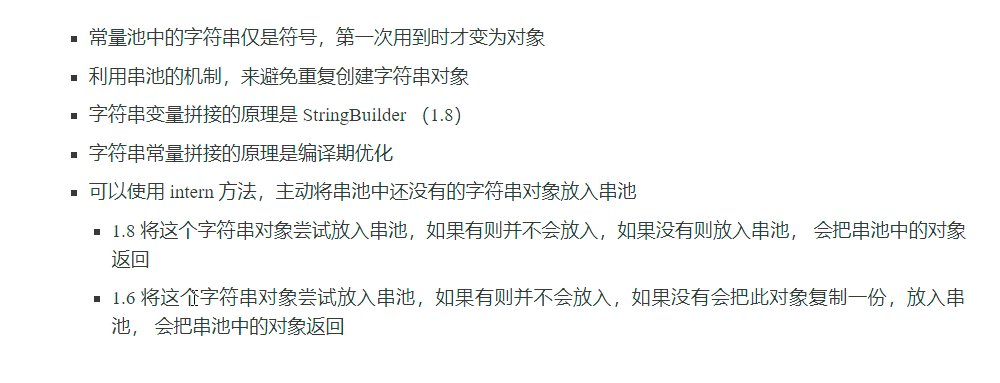


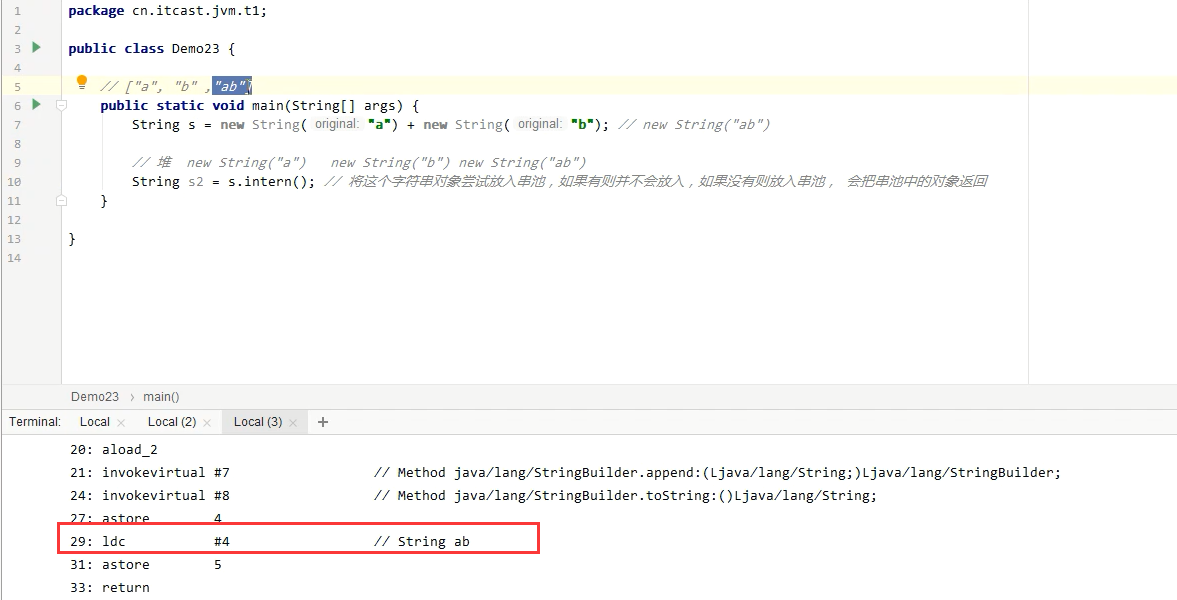
### StringTable













1.8环境，s==”ab” true;

1.6环境，s==”ab” false;



