拉勾教育
—互联网人实战大学—

# 《Java 性能优化实战 21 讲》

李国 前京东、陌陌高级架构师

— 拉勾教育出品 —

# 19 | 高级进阶：JVM 常见优化参数

# 高级进阶：JVM 常见优化参数

拉勾教育
互联网人实战大学

现在大家用的最多的 Java 版本是 **Java 8**

**CMS** 已经在 Java 14 被正式废除

Java 8 和 Java 11 是目前支持的 LTS 版本

# 高级进阶：JVM 常见优化参数

拉勾教育
互联网人实战大学

查看参数默认

```
java -XX:+PrintFlagsFinal -XX:+UseG1GC 2>&1 | grep UseAdaptiveSizePolicy
```

# 高级进阶：JVM 常见优化参数



JVM 默认使用的是并行收集器

```
# java -XX:+PrintCommandLineFlags -version
-XX:InitialHeapSize=127905216 -XX:MaxHeapSize=2046483456 -
XX:+PrintCommandLineFlags -XX:+UseCompressedClassPointers -
XX:+UseCompressedOops -XX:+UseParallelGC
openjdk version "1.8.0_41"
OpenJDK Runtime Environment (build 1.8.0_41-b04)
OpenJDK 64-Bit Server VM (build 25.40-b25, mixed mode)
```

# 高级进阶：JVM 常见优化参数

拉勾教育
互联网人实战大学

ElasticSearch（简称 ES）是一个高性能的开源分布式搜索引擎

ES 是基于 Java 语言开发的，在 conf 目录下，有一个 jvm.options 文件

JVM 的配置就放在这里

# 堆空间的配置



ES 对于堆空间大小的配置

```
-Xms1g

-Xmx1g
```

# 堆空间的配置



配置文件中还有 AlwaysPreTouch 参数

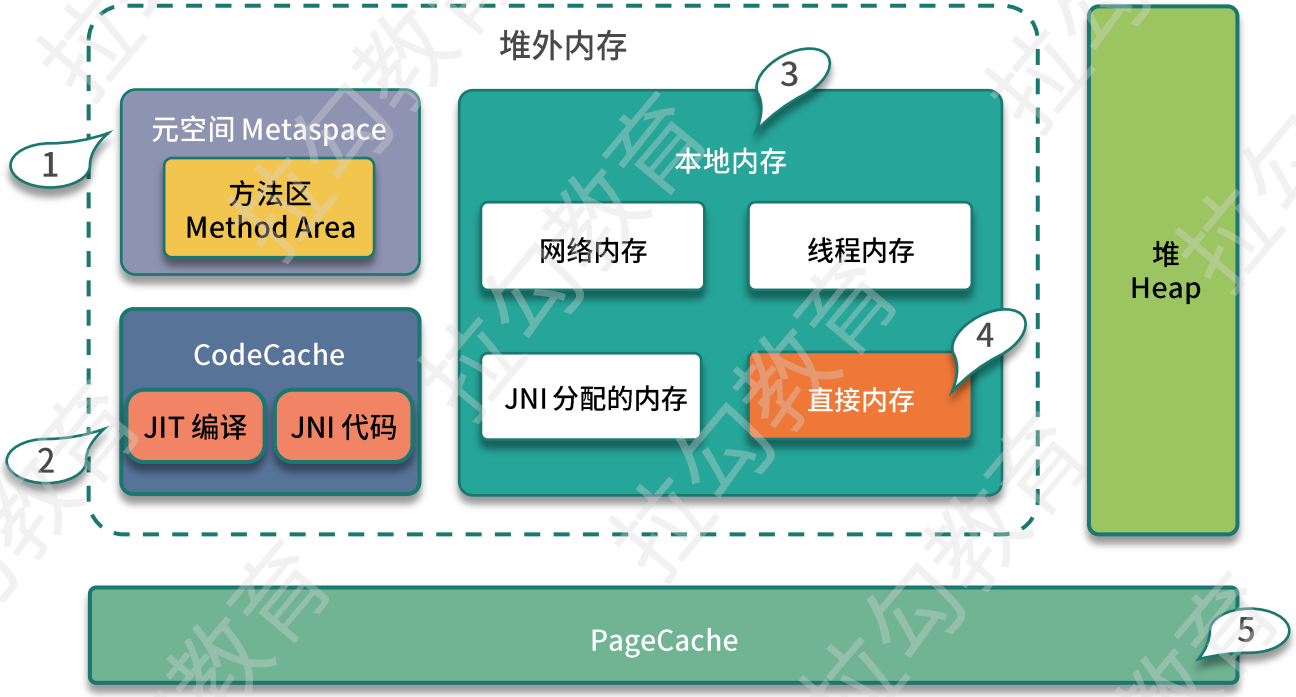
```
-XX:+AlwaysPreTouch
```

# 堆空间的配置



# 堆空间的配置

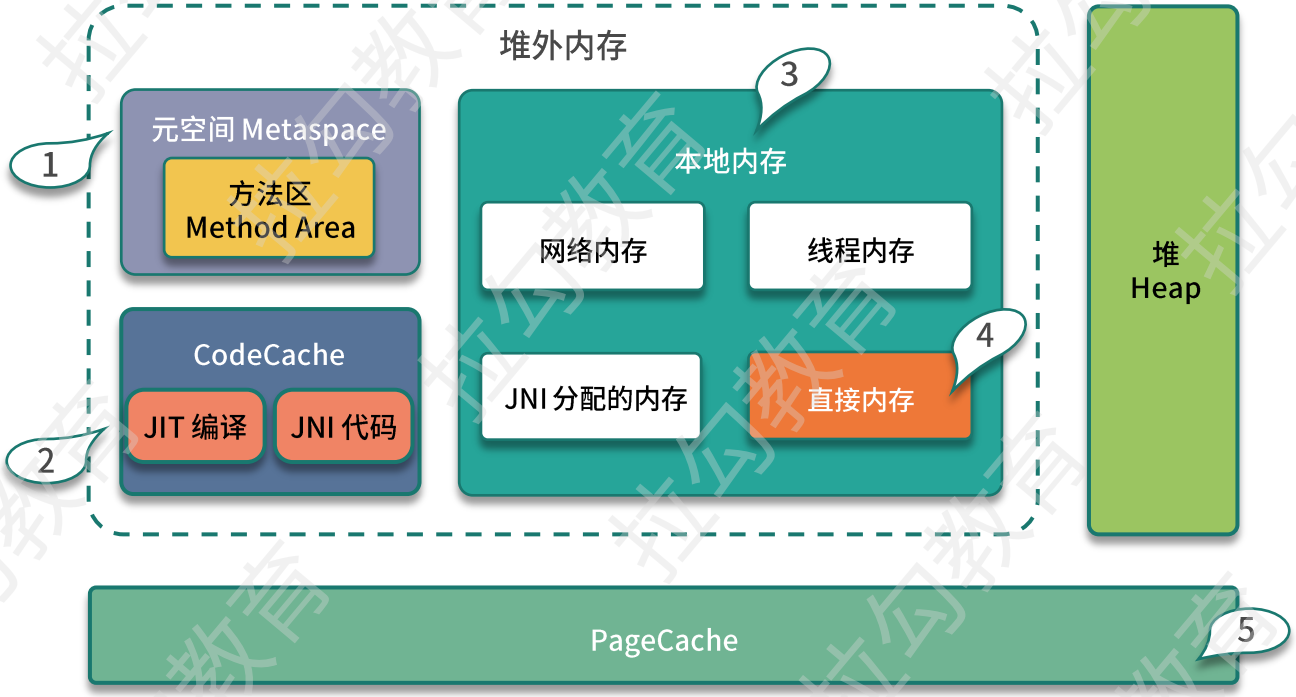





**元空间：** 参数 -XX:MaxMetaspaceSize 和 -XX:MetaspaceSize 分别指定了元空间的最大内存和初始化内存

# 堆空间的配置



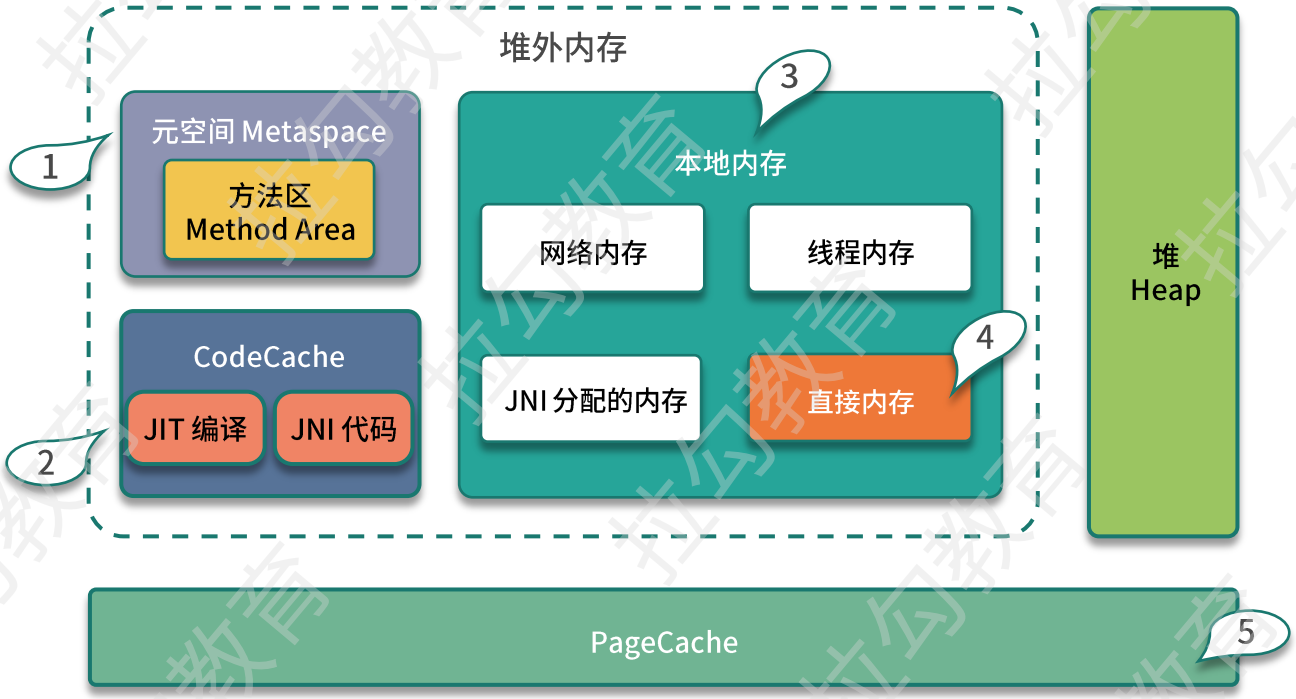



**JIT编译后代码存放**

-XX:ReservedCodeCacheSize

# 堆空间的配置

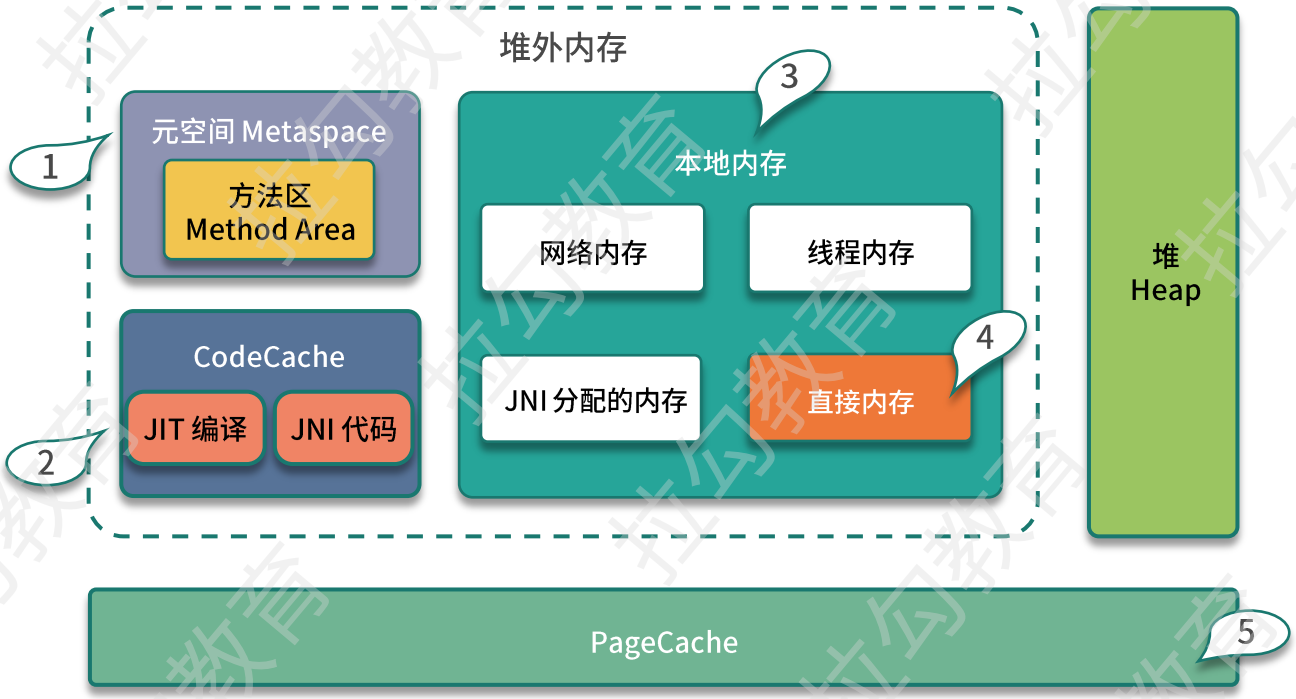





**本地内存**是一些其他 attch 在 JVM 进程上的内存区域的统称

# 堆空间的配置



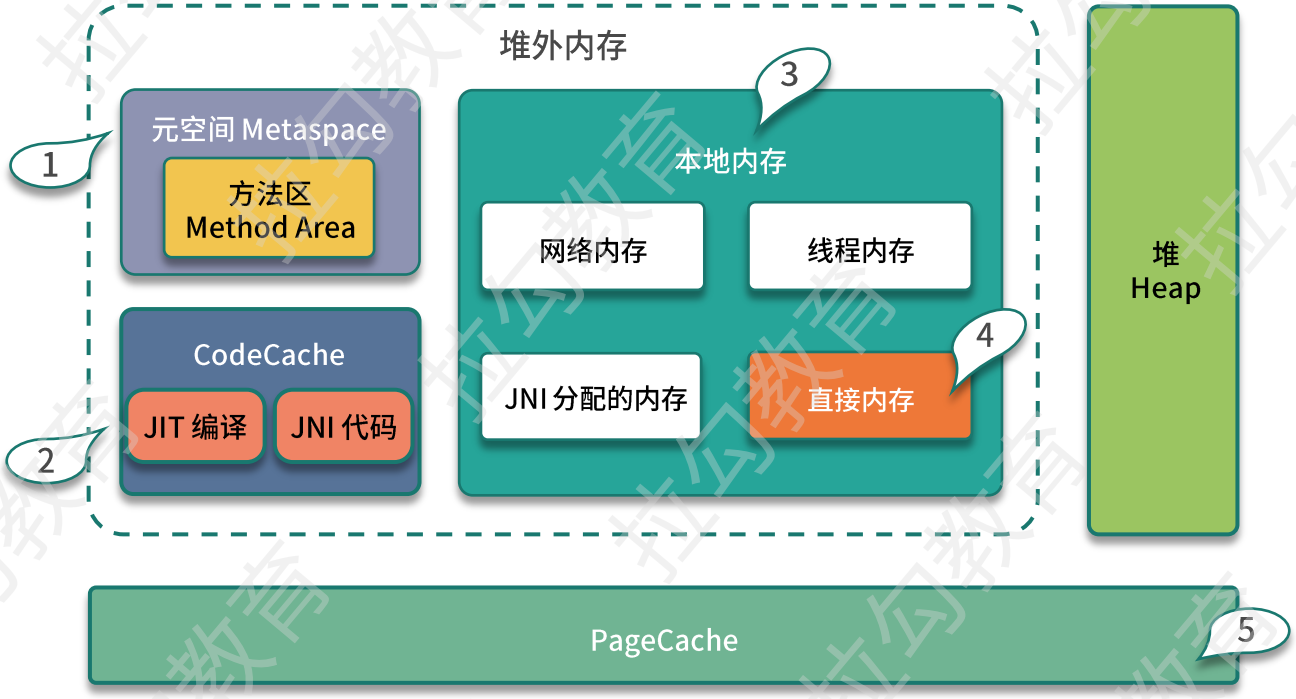



**直接内存**是本地内存中唯一可以使用参数来限制大小的区域

使用参数 -XX:MaxDirectMemorySize，即可设定 ByteBuffer 类所申请的内存上限

# 堆空间的配置






**JNI 内存**指这部分代码所 malloc 的具体内存

# 日志参数配置



ES 的日志参数配置

```
8:-XX:+PrintGCDetails
8:-XX:+PrintGCDateStamps
8:-XX:+PrintTenuringDistribution
8:-XX:+PrintGCApplicationStoppedTime
8:-Xloggc:logs/gc.log
8:-XX:+UseGCLogFileRotation
8:-XX:NumberOfGCLogFiles=32
8:-XX:GCLogFileSize=64m

9-:-
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,t
ags:filecount=32,filesize=64m
```

# 日志参数配置



ES 的日志参数配置

```
8:-XX:+PrintGCDetails
8:-XX:+PrintGCDateStamps
8:-XX:+PrintTenuringDistribution
8:-XX:+PrintGCApplicationStoppedTime
8:-Xloggc:logs/gc.log
8:-XX:+UseGCLogFileRotation
8:-XX:NumberOfGCLogFiles=32
8:-XX:GCLogFileSize=64m

9-:-
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,t
ags:filecount=32,filesize=64m
```

# 日志参数配置



ES 的日志参数配置

```
8:-XX:+PrintGCDetails
8:-XX:+PrintGCDateStamps
8:-XX:+PrintTenuringDistribution
8:-XX:+PrintGCApplicationStoppedTime
8:-Xloggc:logs/gc.log
8:-XX:+UseGCLogFileRotation
8:-XX:NumberOfGCLogFiles=32
8:-XX:GCLogFileSize=64m

9-:-
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,t
ags:filecount=32,filesize=64m
```

# 日志参数配置



ES 的日志参数配置

```
8:-XX:+PrintGCDetails
8:-XX:+PrintGCDateStamps
8:-XX:+PrintTenuringDistribution
8:-XX:+PrintGCApplicationStoppedTime
8:-Xloggc:logs/gc.log
8:-XX:+UseGCLogFileRotation
8:-XX:NumberOfGCLogFiles=32
8:-XX:GCLogFileSize=64m

9-:-
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,tags:filecount=32,filesize=64m
```

# 日志参数配置



ES 的日志参数配置

```
8:-XX:+PrintGCDetails
8:-XX:+PrintGCDateStamps
8:-XX:+PrintTenuringDistribution
8:-XX:+PrintGCApplicationStoppedTime
8:-Xloggc:logs/gc.log
8:-XX:+UseGCLogFileRotation
8:-XX:NumberOfGCLogFiles=32
8:-XX:GCLogFileSize=64m

9-:-
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,t
ags:filecount=32,filesize=64m
```

# 日志参数配置



```
9-:-
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctim
e,pid,tags:filecount=32,filesize=64m
```

# 日志参数配置



ES 在异常情况下的配置参数

```
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=data
-XX:ErrorFile=logs/hs_err_pid%p.log
```

# 垃圾回收器配置



ES 默认使用 CMS 垃圾回收器，它有以下三行主要的配置

```
-XX:+UseConcMarkSweepGC
-XX:CMSInitiatingOccupancyFraction=75
-XX:+UseCMSInitiatingOccupancyOnly
```

# 垃圾回收器配置



ES 默认使用 CMS 垃圾回收器，它有以下三行主要的配置

```
-XX:+UseConcMarkSweepGC
-XX:CMSInitiatingOccupancyFraction=75
-XX:+UseCMSInitiatingOccupancyOnly
```

# 垃圾回收器配置



ES 默认使用 CMS 垃圾回收器，它有以下三行主要的配置

```
-XX:+UseConcMarkSweepGC
-XX:CMSInitiatingOccupancyFraction=75
-XX:+UseCMSInitiatingOccupancyOnly
```

# 垃圾回收器配置

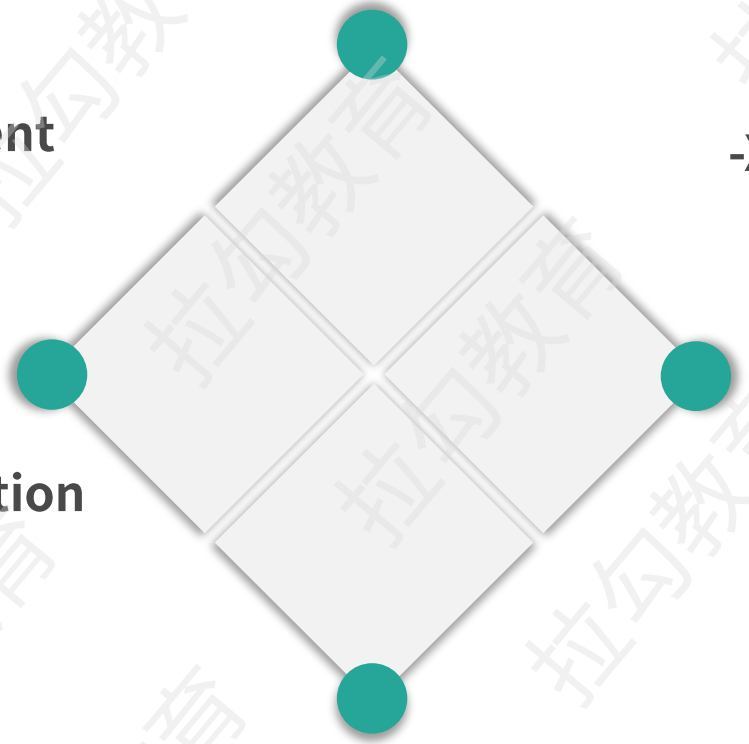
拉勾教育
互联网人实战大学

## -XX:ExplicitGCInvokesConcurrent

当代码里显示的调用了 System.gc()
使用这个参数开始并行 FullGC

## -XX:CMSFullGCsBeforeCompaction

默认为 0，就是每次 FullGC 都对老
年代进行碎片整理压缩



## -XX:CMSScavengeBeforeRemark

开启或关闭在 CMS 重新标记阶段之
前的清除（YGC）尝试

## -XX:+ParallelRefProcEnabled

并行处理 Reference，以加快处理
速度，缩短耗时

# 参数 MaxGCPauseMillis

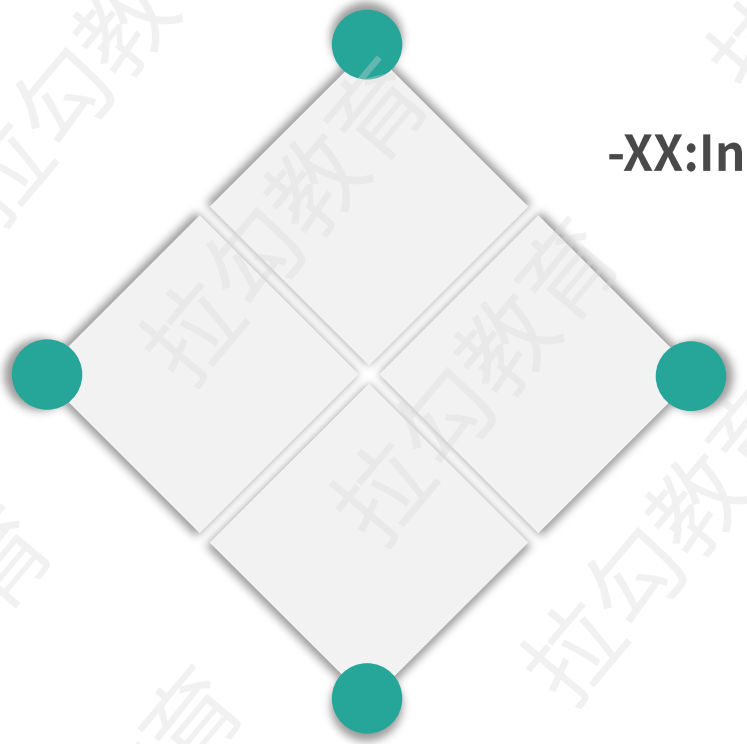
拉勾教育
互联网人实战大学

### -XX:MaxGCPauseMillis

设置目标停顿时间，G1 会尽力达成

### -XX:G1HeapRegionSize

设置小堆区大小。这个值为 2 的次幂
不要太大，也不要太小



### -XX:InitiatingHeapOccupancyPercent

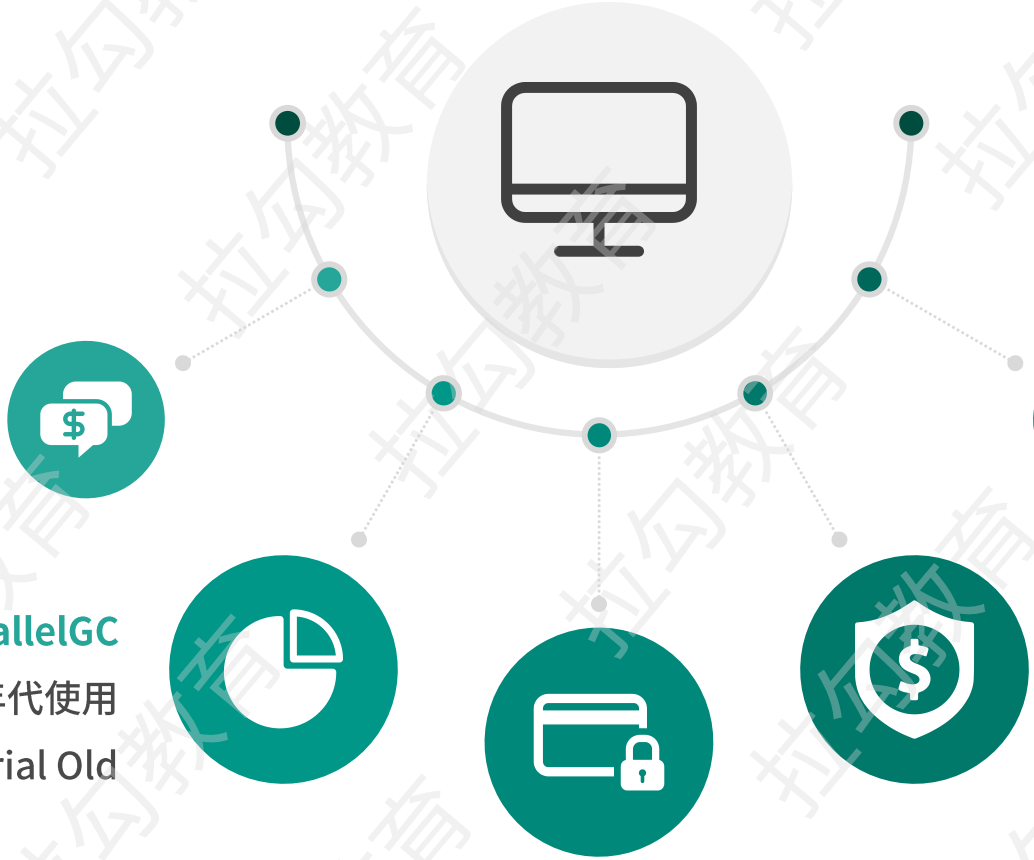
当整个堆内存使用达到一定比例（默认
是45%），并发标记阶段就会被启动

### -XX:ConcGCThreads

并发垃圾收集器使用的线程数量
默认值随 JVM 运行的平台不同而不同

# 垃圾回收器配置





**-XX:+UseSerialGC**
年轻代和老年代都用串行收集器

**-XX:+UseParallelGC**
年轻代使用 ParallerGC，老年代使用 Serial Old

**-XX:+UseParallelOldGC**
新生代和老年代都使用并行收集器

**-XX:+UseZGC**
使用 ZGC 垃圾回收器

**-XX:+UseG1GC**
使用 G1 垃圾回收器

# 额外配置



```
-Xss1m
```

-Xss 设置每个 Java 虚拟机栈的容量为 1MB

# 额外配置



把 - 换成 +，可以减少异常栈的输出

```
-XX:-OmitStackTraceInFastThrow
```

# 额外配置

拉勾教育
互联网人实战大学

```
-Djava.awt.headless=true
```

Headless 模式是系统的一种配置模式

# 额外配置



```
9-:-Djava.locale.providers=COMPAT
-Dfile.encoding=UTF-8
-Des.networkaddress.cache.ttl=60
-Des.networkaddress.cache.negative.ttl=10
-Dio.netty.noUnsafe=true
-Dio.netty.noKeySetOptimization=true
-Dio.netty.recycler.maxCapacityPerThread=0
-Dlog4j.shutdownHookEnabled=false
-Dlog4j2.disable.jmx=true
-Djava.io.tmpdir=${ES_TMPDIR}
-Djna.nosys=true
```

# 其他调优



**-Xmn**

年轻代大小，默认年轻代占堆大小的 1/3

**-XX:SurvivorRatio**

默认值为 8，表示伊甸区和幸存区的比例

# 其他调优



**-XX:MaxTenuringThreshold**

这个值在 CMS 下默认为 6，G1 下默认为 15

对象的年龄分布可以使用 -XX:+PrintTenuringDistribution 打印

**PretenureSizeThreshold**

超过一定大小的对象，将直接在老年代分配

# 练习：cassandra 的参数配置

拉勾教育
互联网人实战大学

大家可以拿 cassandra 的配置文件分析一下

cassandra 是一个高速的列存数据库，使用 gossip 进行集群维护

它的 JVM 参数配置同样在 jvm.options 中

Next：20 | 《SpringBoot 服务性能优化》

# 拉勾教育

—互联网人实战大学—


关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U