> 本文由 JavaGuide 翻译自 https://www.baeldung.com/jvm-parameters，并对文章进行了大量的完善补充。翻译不易，如需转载请注明出处为： 作者： 。

1.概述

在本篇文章中，你将掌握最常用的 JVM 参数配置。如果对于下面提到了一些概念比如堆、

2.堆内存相关

> Java 虚拟机所管理的内存中最大的一块，Java 堆是所有线程共享的一块内存区域，在虚拟机启动时创建。**此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例以及数组都在这里分配内存。**

2.1.显式指定堆内存-Xms和-Xmx

与性能有关的最常见实践之一是根据应用程序要求初始化堆内存。如果我们需要指定最小和最大堆大小（推荐显示指定大小），以下参数可以帮助你实现：

```
-Xms<heap size>[unit]
-Xmx<heap size>[unit]
```

- **heap size** 表示要初始化内存的具体大小。
- **unit** 表示要初始化内存的单位。单位为***“ g”*** (GB) 、**“*m*”**（MB）、**“*k*”**（KB）。

举个栗子🌰，如果我们要为JVM分配最小2 GB和最大5 GB的堆内存大小，我们的参数应该这样来写：

```
-Xms2G -Xmx5G
```

2.2.显式新生代内存(Young Generation)

根据Oracle官方文档，在堆总可用内存配置完成之后，第二大影响因素是为 `Young Generation` 在堆内存所占的比例。默认情况下，YG 的最小大小为 1310 *MB*，最大大小为*无限制*。

一共有两种指定 新生代内存(Young Generation)大小的方法：

1.通过`-XX:NewSize`和`-XX:MaxNewSize`指定

```
-XX:NewSize=<young size>[unit]
-XX:MaxNewSize=<young size>[unit]
```

举个栗子🌰，如果我们要为 新生代分配 最小256m 的内存，最大 1024m的内存我们的参数应该这样来写：

```
-XX:NewSize=256m
-XX:MaxNewSize=1024m
```

2.通过`-Xmn<young size>[unit]`指定

举个栗子🌰，如果我们要为 新生代分配256m的内存（NewSize与MaxNewSize设为一致），我们的参数应该这样来写：

```
-Xmn256m
```

GC 调优策略中很重要的一条经验总结是这样说的：

> 将新对象预留在新生代，由于 Full GC 的成本远高于 Minor GC，因此尽可能将对象分配在新生代是明智的做法，实际项目中根据 GC 日志分析新生代空间大小分配是否合理，适当通过"-Xmn"命令调节新生代大小，最大限度降低新对象直接进入老年代的情况。

另外，你还可以通过**`-XX:NewRatio=<int>`**来设置新生代和老年代内存的比值。

比如下面的参数就是设置新生代（包括Eden和两个Survivor区）与老年代的比值为1。也就是说：新生代与老年代所占比值为1：1，新生代占整个堆栈的 1/2。

```
-XX:NewRatio=1
```

2.3.显式指定永久代/元空间的大小

从Java 8开始，如果我们没有指定 Metaspace 的大小，随着更多类的创建，虚拟机会耗尽所有可用的系统内存（永久代并不会出现这种情况）。

JDK 1.8 之前永久代还没被彻底移除的时候通常通过下面这些参数来调节方法区大小

```
-XX:PermSize=N //方法区 (永久代) 初始大小
-XX:MaxPermSize=N //方法区 (永久代) 最大大小,超过这个值将会抛出 OutOfMemoryError 异常:java.lang.OutOfMemoryError: PermGen
```

相对而言，垃圾收集行为在这个区域是比较少出现的，但并非数据进入方法区后就"永久存在"了。

JDK 1.8 的时候，方法区（HotSpot 的永久代）被彻底移除了（JDK1.7 就已经开始了），取而代之是元空间，元空间使用的是本地内存。

下面是一些常用参数：

```
-XX:MetaspaceSize=N //设置 Metaspace 的初始（和最小大小）
-XX:MaxMetaspaceSize=N //设置 Metaspace 的最大大小，如果不指定大小的话，随着更多类的创建，虚拟机会耗尽所有可用的系统内存。
```

3.垃圾收集相关

3.1.垃圾回收器

为了提高应用程序的稳定性，选择正确的垃圾收集算法至关重要。

JVM具有四种类型的*GC*实现：

- 串行垃圾收集器
- 并行垃圾收集器
- CMS垃圾收集器
- G1垃圾收集器

可以使用以下参数声明这些实现：

```
-XX:+UseSerialGC
-XX:+UseParallelGC
-XX:+USeParNewGC
-XX:+UseG1GC
```

有关*垃圾回收*实施的更多详细信息，请参见此处。

3.2.GC记录

为了严格监控应用程序的运行状况，我们应该始终检查JVM的*垃圾回收*性能。最简单的方法是以人类可读的格式记录*GC*活动。

使用以下参数，我们可以记录*GC*活动：

```
-XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=< number of log files >
-XX:GCLogFileSize=< file size >[ unit ]
-Xloggc:/path/to/gc.log
```

推荐阅读

- CMS GC 默认新生代是多大?
- CMS GC启动参数优化配置
- 从实际案例聊聊Java应用的GC优化-美团技术团队
- JVM性能调优详解 (2019-11-11)
- JVM参数使用手册