java 命令参数详解

2019-09-29 | java | 0 | 136

`java` 命令用于启动 java 应用：它首先会启动 java 运行时环境（JRE），然后加载指定的类，调用类的 `main()` 方法。`main()` 方法必须定义为 `public` 和 `static` 的，并且不返回任何值，参数是 String 类型的数组，该方法的形式如下：

```
1   public static void main(String[] args)
```

在通过 `java` 命令启动应用时，有一系列的可选参数，使用时需根据应用场景选择合适的参数。本文内容主要来自于 Mac OS 下 `java` 命令的 man page。

语法

`java` 命令支持两种启动方式：

1. 一种是指定要执行的 java 类

```
1   java [options] classname [args]
```

2. 另一种是指定要执行的 jar 包

```
1   java [options] -jar filename [args]
```

其中每一项的含义如下：

- options 是可选项，选项之间使用空格分割
- classname 是要加载的类的名字
- filename 是要执行的 java 压缩文件（JAR）的名字，需要和 `-jar` 搭配使用
- args 是传递给 `main()` 方法的参数，使用空格分割

选项

`java` 命令支持一系列选项，它们可以分为如下几类：

- 标准选项
- 非标准选项
- 运行时高级选项
- JIT 编译器高级选项
- 可服务性高级选项
- 垃圾回收器高级选项

所有 java 虚拟机（JVM）都要保证支持标准选项，这些选项用于通用类型的操作，例如检查 JRE 版本、设置 class path 等。

非标准选项以 `-X` 开头，是适用于 java HotSpot 虚拟机的通用选项，其他虚拟机不保证支持，并且以后还可能会变化。

高级选项以 `-XX` 开头，是开发者选项，用于调节 Java HotSpot 虚拟机的特定功能，通常对系统有特定的要求，并且可能还需要访问系统配置参数的特权。并非所有的虚拟机都支持的这些选项，以后还可能会变化。

Boolean 型选项用于开启或关闭特定功能，这种类型的选项不需要参数。对于 `-XX` 选项来说，`-XX:+OptionName` 表示开启功能，`-XX:-OptionName` 表示关闭功能。

标准选项

-agentlib:libname[=options]

用于加载指定的本地代理库，选项（options）之间使用逗号（,）分隔。

例如指定 `-agentlib:foo` 参数后，JVM 会尝试到 `LD_LIBRARY_PATH` 系统变量指定的路径（OS X 系统下变量是 `DYLD_LIBRARY_PATH`）下加载 `libfoo.so` 类库。

如下的示例展示了如何加载 HPROF（heap profiling tool）库，并且每隔 20 ms 获取一次 CPU 信息，栈深度为 3：

```
1   -agentlib:hprof=cpu=samples,interval=20,depth=3
```

这一个示例展示了如何加载 JDWP（Java Debug Wire Protocol），并且在 8000 端口监听 socket 连接，在 main 类加载前暂停：

```
1  -agentlib:jdwp=transport=dt_socket,server=y,address=8000
```

更多本地代理库相关的信息请参考

- `java.lang.instrument` 包的说明

  http://docs.oracle.com/javase/8/docs/api/java/lang/instrument/package-summary.html

- JVM Tools Interface 手册中「本地代理命令行选项」部分

  http://docs.oracle.com/javase/8/docs/platform/jvmti/jvmti.html#starting

-agentpath:pathname[=options]

在指定的绝对路径下加载本地代理库，和 `-agentlib` 的区别是使用的全路径和类库的文件名

-client

选择 Java HotSpot Client 虚拟机。64 位的 JDK（Java SE Development Kit）会忽略该选项，并且使用 Server 虚拟机。

-server

选择 Java HotSpot Server 虚拟机。64 位的 JDK仅支持 Server 虚拟机，所以该选项是默认值。

要了解不同类型机器默认选择的 JVM，可查看如下页面

http://docs.oracle.com/javase/8/docs/technotes/guides/vm/server-class.html

-Dproperty=value

用于设置系统属性（property）值。例如： `-Dfoo="foo bar"`

-d32

在 32 位环境下运行程序，如果未安装 32 位环境或者不支持，系统会报告一个错误。

-d64

在 64 位环境下运行程序。 `-server` 选项会隐式地包含 `-d64` 选项。

-jar filename

执行封装在 JAR 文件中的程序。JAR 文件需包含 manifest，其中一行格式为 `Main-Class:classname`，指定了一个包含 `public static void main(String[] args)` 方法的类，作为该程序的启动点。

当使用了 `-jar` 选项时，所有的用户类都从 JAR 文件中读取，忽略其他的 class path 设置。

-showversion

显示版本信息并且继续执行程序。

-version

显示版本信息然后退出程序。

-splash:imgname

显示指定的闪屏图片。例如：要在启动程序时显示 `image` 目录下的 `splash.gif` 图片，可使用如下参数：

```
1   -splash:images/splash.gif
```

-verbose:class

显示每一次加载类的信息。

-verbose:gc

显示每一次垃圾回收（GC）事件的信息。

-verbose:jni

显示使用的本地方法信息和其他的 JNI（Java Native Interface）活动信息。

非标准选项

这些是 Java HotSpot 虚拟机的通用目的选项。

-X

显示所有适用于 `-X` 选项的帮助信息。

-Xbatch

禁用后台编译，该参数等价于 `-XX:-BackgroundCompilation` 。默认情况下，JVM 使用一个后台任务进行编译，在编译完成前使用解释模式执行方法。加上该参数后将会在前台编译所有的方法。

-Xbootclasspath:path

设置 boot class 的路径， `path` 可以是目录、JAR 文件、ZIP 压缩包，中间使用 `:` 分割。不要使用该选项覆盖 `rt.jar` 里的类，因为这会违反「JRE binary code license」。另外 `-Xbootclasspath/a:path` 用于添加路径到默认路径的后面， `-Xbootclasspath/p:path` 用于添加路径到默认路径的前面。

-Xcheck:jni

对 JNI（Java Native Interface）函数执行额外的检查。具体来讲，在处理 JNI 请求前，它会检查传递给 JNI 函数的参数和运行时环境数据。如果检查到任何的不合法数据，JVM 会抛出不可恢复的错误并结束执行。该参数会降低性能。

-Xcomp

强制在方法的第一次调用时进行编译。默认情况下，Client 虚拟机（使用 -client 参数）会先执行 1000 次解释执行，Server 虚拟机（-server）是 10000 次，以便获取到足够的信息用于编译。还可以使用 `-XX:CompileThreshold` 指定编译前解释执行的次数。

-Xint

仅使用解释模式执行程序。所有的代码都在解释模式下执行，这将无法体验到 JIT（just in time）编译器带来的性能提升。

-Xinternalversion

显示比 `-version` 更详细的版本信息。

-Xloggc:filename

重定向 GC 事件输出信息到指定的文件。该参数和 `-verbose:gc` 输出的信息类似，如果同时指定则会覆盖后者。使用示例：

```
1   -Xloggc:garbage-collection.log
```

-Xmaxjitcodesize=size

指定最大的代码缓存，用于保存 JIT 编译后的代码，该选项和 `-XX:ReservedCodeCacheSize` 相等。默认的大小是 240 MB。如果使用 `-XX:-TieredCompilation` 禁用了分层编译，则默认大小是 48M。

-Xmixed

使用混合模式执行。热点代码使用编译模式，其他使用解释模式。

-Xmnsize

设置年轻代的初始容量和最大容量，默认单位是字节。如下是一些示例：

```
-Xmn256m
-Xmn262144k
-Xmn268435456
```

此外，你也可以使用 `-XX:NewSize` 指定初始容量，使用 `-XX:MaxNewSize` 指定最大容量。

-Xmssize

设置堆的初始容量，默认单位是字节。必须是 1024 的倍数且大于 1 MB。如下是一些示例：

```
-Xms6291456
-Xms6144k
-Xms6m
```

如果未设置该选项，则初始堆大小为老年代和年轻代分批的容量之和。

-Xmxsize

设置堆可分配的最大容量，默认单位是字节，该选项和 `-XX:MaxHeapSize` 相等。必须是 1024 的倍数且大于 2 MB。对于服务器来说，通常将 `-Xms` 和 `-Xmx` 设置为相同值。如下是一些示例：

```
-Xmx83886080
-Xmx81920k
-Xmx80m
```

关于参数优化请查看「Java SE HotSpot Virtual Machine Garbage Collection Tuning Guide」。

-Xnoclassgc

禁用类的 GC。这可以节省一些 GC 时间，减少应用程序被打断的次数。开启该参数后，GC 期间不会处理类对象，类对象会一直存活，这将导致更多的永久内存占用，因此需小心使用。

-Xprof

分析正在执行的程序，把分析数据发送到标准输出。该参数可在开发时使用，不要用到生产环境。

-Xrs

减少 JVM 对操作系统信号的使用。

使用 shutdown hook 可以在应用程序退出（包括 JVM 异常退出）时按顺序执行一些清理工作（比如关闭数据库连接）。JVM 通过捕获系统信号来实现异常退出时的 shutdown hook，它使用 `SIGHUP`、`SIGINT` 和 `SIGTERM` 来初始化正在运行的 shutdown hook。

JVM 使用了类似的机制实现了 dump 线程栈的功能，它使用的是 `SIGQUIT` 信号。

嵌入到 JVM 的程序经常需要捕获 `SIGINT` 或 `SIGTERM` 之类的信号，这可能会干扰 JVM 信号处理的 handler，该选项可用于解决此类问题。

当使用该选项时，JVM 不会修改 `SIGINT`、`SIGHUP`、`SIGTERM` 和 `SIGQUIT` 信号的掩码（mask），也不会设置用于处理这些信号的 handler。因此使用该参数后：

- `SIGQUIT` 将不会导致线程 dump
- 用户代码负责触发 shutdown hook，例如在退出前调用 `System.exit()`

-Xsssize

设置线程栈的大小，默认单位为字节，该选项与 `-XX:ThreadStackSize` 相等。如下是不同平台的默认值：

- Linux/ARM (32-bit): 320 KB
- Linux/i386 (32-bit): 320 KB
- Linux/x64 (64-bit): 1024 KB
- OS X (64-bit): 1024 KB
- Oracle Solaris/i386 (32-bit): 320 KB
- Oracle Solaris/x64 (64-bit): 1024 KB

如下是将栈大小设置为 1024 KB的示例：

```
1   -Xss1m
2   -Xss1024k
3   -Xss1048576
```

运行时高级选项

这些选项用于控制 Java HotSpot 虚拟机运行时的行为。

-XX:+PrintFlagsInitial

打印 JVM flag 参数的初始值。

-XX:+PrintCommandLineFlags

打印出现在命令行中的 JVM flag 参数。

-XX:+DisableAttachMechanism

启用该选项后会禁用 attach 到 JVM 的机制，使得无法使用 `jcmd`、 `jstack`、 `jmap` 、 `jinfo` 等工具。

-XX:MaxDirectMemorySize=size

设置 NIO 可分配的 direct-buffer 的最大容量，默认单位为字节。

-XX:-UseBiasedLocking

禁用偏向锁。该选项默认开启（-XX:+UseBiasedLocking）。关于偏向锁的设置请参考「Java Tuning White Paper」

-XX:-UseCompressedOops

禁用压缩指针。该选项默认开启。压缩指针用于堆内存小于 32 GB 的情况。当开启压缩指针时，使用 32 位的偏移（代替 64 位的指针）表示对象引用。若要在堆大小超过 32 GB 时使用压缩指针，需使用 `-XX:ObjectAlignmentInBytes` 选项。

JIT 编译器高级选项

这些选项用于控制 JIT（just-in-time）编译器在 Java HotSpot 虚拟机中的表现。

-XX:CompileOnly=methods

设置仅编译指定的方法。需使用全类名指定要编译的方法。例如只编译 `String` 类的 `length()` 方法和 `List` 类的 `size()` 方法，可使用如下配置：

```
1   -XX:CompileOnly=java/lang/String.length,java/util/List.size
```

为方便使用，它同时还支持 `-XX:+PrintCompilation` 和 `-XX:+LogCompilation` 的输出格式，例如：

```
1   -XX:CompileOnly=java.lang.String::length,java.util.List::size
```

-XX:+DoEscapeAnalysis

启用逃逸分析。该选项默认开启。

-XX:InitialCodeCacheSize=size

设置初始代码缓存大小。

-XX:+Inline

启用方法内联。

-XX:InlineSmallCode=size

针对已编译方法，设置应该进行内联的方法的最大大小。默认单位为字节。只有已编译的方法且大小小于该值的才会被内联。它的默认值为 1000 字节

```
1   -XX:InlineSmallCode=1000
```

-XX:MaxInlineSize=size

设置要内联的方法的最大字节码大小。默认单位为字节。它的默认值为 35 字节

```
1   -XX:MaxInlineSize=35
```

可服务化高级选项

这些选项提供了收集系统信息和执行扩展调试的能力。

-XX:+HeapDumpOnOutOfMemory

当 JVM 抛出 `java.lang.OutOfMemoryError` 异常时，使用 HPROF（the heap profiler）将 java 堆 dump 到当前目录。该选项默认禁用。

-XX:HeapDumpPath=path

与 `-XX:+HeapDumpOnOutOfMemoryError` 结合起来使用，用来指定 dump 文件的路径和文件名。如下是一个示例，其中 `%p` 表示当前进程的 ID

```
1   -XX:HeapDumpPath=./java_pid%p.hprof
```

垃圾回收器高级选项

这些选项用来设置 Java HotSpot 使用的垃圾回收器。

-XX:CMSInitiatingOccupancyFraction=percent

用于设置老年代使用比例到达多少（0 - 100）时执行 CMS 垃圾回收。默认值是 -1。任何负值都意味着使用 `-XX:CMSTriggerRatio` 作为初始数值。如下表示设置为 20%

```
1   -XX:CMSInitiatingOccupancyFraction=20
```

-XX:CMSTriggerRatio=percent

设置一个百分比，表示当 `-XX:MinHeapFreeRatio` 指定的值到达多少时执行 CMS 垃圾回收。默认设置为 80%。如下示例将占比设置为 75%

```
1   -XX:CMSTriggerRatio=75
```

-XX:ConcGCThreads=threads

设置并发 GC 使用的线程数。例如设置为 2

```
1   -XX:ConcGCThreads=2
```

-XX:+DisableExplicitGC

开启该选项会禁用对 `System.gc()` 的处理。该选项默认禁用，表示会处理 `System.gc()` 调用。若开启该选项，JVM 仍会在必要是执行 GC。

-XX:G1HeapRegionSize=size

当使用 G1（garbage-first）收集器时，使用该选项设置 java 堆划分后的 region 大小。可设置为 1 MB 到 32 MB 之间。默认 region 大小依赖于堆的大小。如下示例将 region 设置为 16 MB

```
1   -XX:G1HeapRegionSize=16m
```

-XX:MaxGCPauseMillis=time

设置最大 GC 暂停时间（单位为毫秒），JVM 将尽最大努力实现它。如下表示设置为 500 ms

```
-XX:MaxGCPauseMillis=500
```

-XX:MaxMetaspaceSize=size

设置可分配给 class 元数据（metadata）的最大本地内存。

```
-XX:MaxMetaspaceSize=256m
```

-XX:MaxNewSize=size

设置最大的新生代大小。

-XX:MaxTenuringThreshold=threshold

设置用于自适应 GC 进入老年代时的最大生存时间（经历多少次 GC）阈值。最大可设置为 15。parallel 收集器默认是 15，CMS 收集器默认是 6。

```
-XX:MaxTenuringThreshold=10
```

-XX:MetaspaceSize=size

设置 class 元数据第一次占用空间到达多少时触发 GC。垃圾回收器会根据 meta 空间的使用情况动态调整该阈值

```
-XX:MinHeapFreeRatio=25
```

-XX:NewRatio=ratio

设置年轻代和老年代的比值。默认值为 2。如下示例将 young/old 设置为 1

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

-XX:NewSize=size

设置年轻代的初始占用空间。

-XX:SurvivorRatio=ratio

设置 eden 和 survivor 的比例。默认值为 8。如下示例将 eden/survivor 比例设置为 4


```
-XX:SurvivorRatio=4
```

-XX:+PrintGC

每次 GC 时打印相关信息。该选项默认禁用。

-XX:+PrintGCDetails

每次 GC 时打印相关详情信息。该选项默认禁用。

-XX:TLABSize=size

设置单个 TLAB（a thread-local allocation buffer）的初始大小。如果设置为 0，JVM 会自动选择初始大小。

```
1   -XX:TLABSize=512k
```

-XX:+UseConcMarkSweepGC

使用 CMS 作为老年代的垃圾收集器，同时会自动设置 `-XX:+UseParNewGC` 选项，将 ParNewGC 设置为新生代的垃圾收集器。

-XX:+UseG1GC

使用 G1（garbage-first）垃圾收集器。

-XX:+UseParallelGC

使用 ParallelGC（parallel scavenge garbage collector）垃圾收集器。默认会设置 `-XX:+UseParallelOldGC` 。

-XX:+UseParallelOldGC

使用 ParallelOldGC 垃圾收集器处理 full GC。默认会设置 `-XX:+UseParallelGC` 。

-XX:+UseParNewGC

使用 ParNewGC（parallel threads for collection）作为新生代的垃圾收集器。当设置 `-XX:+UseConcMarkSweepGC` 时会自动启用。

-XX:+UseSerialGC

使用 SerialGC（serial garbage collector）垃圾收集器

-XX:+UseTLAB

在年轻代启用 TLAB（thread-local allocation blocks）。默认开启。

性能调节示例

1. 高吞吐量

```
1  java -d64 -server -XX:+AggressiveOpts -XX:+UseLargePages -Xmn10g  -Xms26g -Xmx
```

2. 低延迟

```
1  java -d64 -XX:+UseG1GC -Xms26g Xmx26g -XX:MaxGCPauseMillis=500 -XX:+PrintGCTim
```

参考

> java 命令的 man page

java

JNI入门之HelloWorld

JNI入门之详细介绍

提交

来发评论吧~

Powered By Valine
v1.4.14

由 Hexo 强力驱动 | 主题 – NexT.Mist