

JVM诊断工具

[零 Reference 6](#_Toc66818885)

[一 JDK Analysis Tool 6](#_Toc66818886)

[1 jps 6](#_Toc66818887)

[1.1 Description 6](#_Toc66818888)

[1.2 Synopsis 6](#_Toc66818889)

[1.3 Output Format of the jps command 7](#_Toc66818890)

[1.4 Example 7](#_Toc66818891)

[2 jstat 8](#_Toc66818892)

[2.1 Description 8](#_Toc66818893)

[2.2 Synopsis 8](#_Toc66818894)

[2.3 Virtual Machine Identifier 8](#_Toc66818895)

[2.4 Output Options and Output 9](#_Toc66818896)

[2.5 Example 14](#_Toc66818897)

[3 jinfo 15](#_Toc66818898)

[3.1 Description 15](#_Toc66818899)

[3.2 Synopsis 16](#_Toc66818900)

[4 jmap 16](#_Toc66818901)

[4.1 Description 16](#_Toc66818902)

[4.2 Synopsis 16](#_Toc66818903)

[4.3 Example 17](#_Toc66818904)

[5 jhat 17](#_Toc66818905)

[6 jstack 17](#_Toc66818906)

[6.1 Description 17](#_Toc66818907)

[6.2 Synopsis 17](#_Toc66818908)

[7 jstatd 17](#_Toc66818909)

[7.1 Description 17](#_Toc66818910)

[7.2 Synopsis 18](#_Toc66818911)

[7.3 Security 18](#_Toc66818912)

[7.4 Example 19](#_Toc66818913)

[8 jcmd 19](#_Toc66818914)

[8.1 Description 19](#_Toc66818915)

[8.2 Synopsis 20](#_Toc66818916)

[8.3 Commands for jcmd 20](#_Toc66818917)

[二 Frequently Execution Instruction 31](#_Toc66818918)

[1 java 31](#_Toc66818919)

[1.1 Description 31](#_Toc66818920)

[1.2 Synopsis 31](#_Toc66818921)

[1.3 Using Source-File Mode to Launch Single-File Source-Code Programs 32](#_Toc66818922)

[1.4 In source-file mode 32](#_Toc66818923)

[1.5 Overview of Java Option 33](#_Toc66818924)

[1.6 Standard Options for Java 34](#_Toc66818925)

[1.7 Extra Options for Java 37](#_Toc66818926)

[1.8 Advanced Options for Java 41](#_Toc66818927)

[1.9 Advanced Runtime Options for Java 42](#_Toc66818928)

[1.10 Advanced JIT Compiler Options for java 42](#_Toc66818929)

[1.11 Advanced Serviceability Options for Java 42](#_Toc66818930)

[1.12 Advanced Garbage Collection Options for Java 42](#_Toc66818931)

[1.13 Obsolete Java Options 42](#_Toc66818932)

[1.14 Using the JDK\_JAVA\_OPTIONS Launcher Environment Variable 42](#_Toc66818933)

[1.15 java Command-Line Argument Files 43](#_Toc66818934)

[1.16 Example 43](#_Toc66818935)

[2 jar 43](#_Toc66818936)

[2.1 Description 43](#_Toc66818937)

[2.2 Synopsis 43](#_Toc66818938)

[2.3 Options 43](#_Toc66818939)

[2.4 Example 45](#_Toc66818940)

[3 javap 45](#_Toc66818941)

[3.1 Description 45](#_Toc66818942)

[3.2 Synopsis 46](#_Toc66818943)

[3.3 Options for javap 46](#_Toc66818944)

[3.4 Example 47](#_Toc66818945)

[三 Other 48](#_Toc66818946)

[1 jaotc 48](#_Toc66818947)

[1.1 Description 48](#_Toc66818948)

[1.2 Synopsis 48](#_Toc66818949)

[1.3 Options 48](#_Toc66818950)

[1.4 Example 49](#_Toc66818951)

[1.5 What is Ahead of Time Compilation: 50](#_Toc66818952)

[2 jarsigner 50](#_Toc66818953)

[2.1 Description 50](#_Toc66818954)

[2.2 Synopsis 50](#_Toc66818955)

[2.3 Options 51](#_Toc66818956)

[2.4 Example 55](#_Toc66818957)

[3 javadoc 55](#_Toc66818958)

[3.1 Description 55](#_Toc66818959)

[3.2 Synopsis 55](#_Toc66818960)

[3.3 Options for javadoc 55](#_Toc66818961)

[3.4 Extended Options 57](#_Toc66818962)

[3.5 Standard doclet Options 58](#_Toc66818963)

[3.6 Additional Options Provided by the Standard Doclet 62](#_Toc66818964)

[3.7 Conformance 63](#_Toc66818965)

[3.8 Example 63](#_Toc66818966)

[4 jconsole 63](#_Toc66818967)

[4.1 Description 63](#_Toc66818968)

[4.2 Synopsis 63](#_Toc66818969)

[5 jdb 64](#_Toc66818970)

[5.1 Description 64](#_Toc66818971)

[5.2 Synopsis 64](#_Toc66818972)

[5.3 Options for the jdb command 64](#_Toc66818973)

[5.4 Operation 65](#_Toc66818974)

[6 jdeprscan 66](#_Toc66818975)

[6.1 Description 66](#_Toc66818976)

[6.2 Synopsis 66](#_Toc66818977)

[6.3 Example 66](#_Toc66818978)

[7 jdeps 67](#_Toc66818979)

[7.1 Description 67](#_Toc66818980)

[7.2 Synopsis 67](#_Toc66818981)

[7.3 Example 67](#_Toc66818982)

[8 jfr 67](#_Toc66818983)

[8.1 Description 67](#_Toc66818984)

[8.2 Synopsis 67](#_Toc66818985)

[8.3 Example 67](#_Toc66818986)

[9 jhsdb 67](#_Toc66818987)

[9.1 Description 67](#_Toc66818988)

[9.2 Synopsis 67](#_Toc66818989)

[9.3 Example 67](#_Toc66818990)

[10 jlink 67](#_Toc66818991)

[10.1 Description 67](#_Toc66818992)

[10.2 Synopsis 67](#_Toc66818993)

[10.3 Example 67](#_Toc66818994)

[11 jmod 67](#_Toc66818995)

[11.1 Description 67](#_Toc66818996)

[11.2 Synopsis 67](#_Toc66818997)

[11.3 Example 67](#_Toc66818998)

[12 jpackage 67](#_Toc66818999)

[12.1 Description 67](#_Toc66819000)

[12.2 Synopsis 67](#_Toc66819001)

[12.3 Example 67](#_Toc66819002)

[13 jrunscript 67](#_Toc66819003)

[13.1 Description 67](#_Toc66819004)

[13.2 Synopsis 67](#_Toc66819005)

[13.3 Example 67](#_Toc66819006)

[14 jshell 67](#_Toc66819007)

[14.1 Description 67](#_Toc66819008)

[14.2 Synopsis 67](#_Toc66819009)

[14.3 Example 67](#_Toc66819010)

[15 keytool 67](#_Toc66819011)

[15.1 Description 68](#_Toc66819012)

[15.2 Synopsis 68](#_Toc66819013)

[15.3 Exampl 68](#_Toc66819014)

[16 rmid 68](#_Toc66819015)

[16.1 Description 68](#_Toc66819016)

[16.2 Synopsis 68](#_Toc66819017)

[16.3 Example 68](#_Toc66819018)

[17 rmiregistry 68](#_Toc66819019)

[17.1 Description 68](#_Toc66819020)

[17.2 Synopsis 68](#_Toc66819021)

[17.3 Example 68](#_Toc66819022)

[18 serialver 68](#_Toc66819023)

[18.1 Description 68](#_Toc66819024)

[18.2 Synopsis 68](#_Toc66819025)

[18.3 Example 68](#_Toc66819026)

[四 OQL(Object Query Language) 68](#_Toc66819027)

[1 Description 68](#_Toc66819028)

[2 BNF(Backus-Naur Form) 68](#_Toc66819029)

[2.1 Description 68](#_Toc66819030)

[五 Eclipse MAT OQL(1.11.0) 68](#_Toc66819031)

[1 Synopsis 68](#_Toc66819032)

[1.1 Document 68](#_Toc66819033)

[1.2 Syntax 69](#_Toc66819034)

[1.3 SELECT 69](#_Toc66819035)

[1.4 FROM 69](#_Toc66819036)

[1.5 WHERE 70](#_Toc66819037)

[1.6 UNION 71](#_Toc66819038)

[2 Property Accessors 71](#_Toc66819039)

[2.1 Accessing fields of the heap object 71](#_Toc66819040)

[2.2 Calling Java Bean properties： 71](#_Toc66819041)

[2.3 Calling Java methods 72](#_Toc66819042)

[2.4 Array Access 72](#_Toc66819043)

[2.5 Built-in OQL function 73](#_Toc66819044)

[3 Simulated in OQL 74](#_Toc66819045)

[4 More Example 79](#_Toc66819046)

[5 BNF for the Object Query Language 79](#_Toc66819047)

[六 JVisualMV OQL 79](#_Toc66819048)

[1 Synopsis 79](#_Toc66819049)

[1.1 Document 79](#_Toc66819050)

[1.2 Syntax 80](#_Toc66819051)

[2 Built-in Object——heap 80](#_Toc66819052)

[2.1 Introduction 80](#_Toc66819053)

[2.2 Example 80](#_Toc66819054)

[3 Functions on individual objects 81](#_Toc66819055)

[3.1 Introduction 81](#_Toc66819056)

[3.2 Example 81](#_Toc66819057)

[4 array/iterator/enumeration manipulation functions 82](#_Toc66819058)

[4.1 Introduction 82](#_Toc66819059)

[4.2 Example 83](#_Toc66819060)

[5 More examples 84](#_Toc66819061)

[七 可视化工具列表 86](#_Toc66819062)

[1 Visual VM 86](#_Toc66819063)

[2 JConsole 86](#_Toc66819064)

[3 Mission Control 86](#_Toc66819065)

[4 JHSDB 86](#_Toc66819066)

[5 Eclipse MAT 86](#_Toc66819067)

# 零 Reference

《深入理解Java虚拟机——JVM高级特性与最佳实践 第三版》

《实战Java虚拟机——JVM故障诊断与性能优化》

https://docs.oracle.com/en/java/javase/index.html

基于 JDK16，排除了文档中针对windows和macOS特定内容

# JDK Analysis Tool

## jps

### Description

* + - 1. The jps command lists the instrumented Java HotSpot VMs on the target system.
      2. The command is limited to reporting information on JVMs for which it has the access permissions.
      3. If the jps command is run without specifying a hostid, then it searches for instrumented JVMs on the local host.
      4. If started with a hostid, then it searches for JVMs on the indicated host, using the specified protocol and port.
      5. A jstatd process is assumed to be running on the target host.
      6. The jps command reports the local JVM identifier, or lvmid, for each instrumented JVM found on the target system.
         1. The lvmid is typically, but not necessarily, the operating system's process identifier for the JVM process.
         2. With no options, the jps command lists each Java application's lvmid followed by the short form of the application's class name or jar file name.
         3. The short form of the class name or JAR file name omits the class's package information or the JAR files path information.
      7. The jps command uses the Java launcher to find the class name and arguments passed to the main method.
      8. If the target JVM is started with a custom launcher, then the class or JAR file name, and the arguments to the main method aren't available.
         1. In this case, the jps command outputs the string Unknown for the class name, or JAR file name, and for the arguments to the main method.
         2. The list of JVMs produced by the jps command can be limited by the permissions granted to the principal running the command.
         3. The command lists only the JVMs for which the principal has access rights as determined by operating system-specific access control mechanisms.

### Synopsis

jps [-q] [-mlvV] [hostid]

* + - 1. Option

|  |  |
| --- | --- |
| -q | Suppresses the output of the class name, JAR file name, and arguments passed to the main method, producing a list of only local JVM identifiers. |
| -mlvV | You can specify any combination of these options.  -m  displays the arguments passed to the main method. The output may be null for embedded JVMs.  -l  displays the full package name for the application's main class or the full path name to the application's JAR file.  -v  displays the arguments passed to the JVM.  -V  suppresses the output of the class name, JAR file name, and arguments passed to the main method, producing a list of only local JVM identifiers. |
|  |  |

* + - 1. Host Identifier

The host identifier, or hostid, is a string that indicates the target system.

The syntax of the hostid string corresponds to the syntax of a URI:

[*protocol*:][[//]*hostname*][:*port*][/*servername*]

|  |  |
| --- | --- |
| protocol | The communications protocol.  If the protocol is omitted and a hostname isn't specified, then the default protocol is a platform-specific, optimized, local protocol.  If the protocol is omitted and a host name is specified, then the default protocol is rmi. |
| hostname | A host name or IP address that indicates the target host.  If you omit the hostname parameter, then the target host is the local host. |
| port | The default port for communicating with the remote server.  If the hostname parameter is omitted or the protocol parameter specifies an optimized, local protocol, then the port parameter is ignored.  Otherwise, treatment of the port parameter is implementation-specific.  For the default rmi protocol, the port parameter indicates the port number for the rmiregistry on the remote host.  If the port parameter is omitted, and the protocol parameter indicates rmi, then the default rmiregistry port (1099) is used. |
| servername | The treatment of this parameter depends on the implementation.  For the optimized, local protocol, this field is ignored.  For the rmi protocol, this parameter is a string that represents the name of the RMI remote object on the remote host.  See the **jstatd** command -n option. |

### Output Format of the jps command

lvmid [[classname|JARfilename|"Unknown"] [arg\*] [jvmarg\*]]

* + - 1. All output tokens are separated by white space.
      2. An arg value that includes embedded white space introduces ambiguity when attempting to map arguments to their actual positional parameters.
      3. It's recommended that you don't write scripts to parse jps output because the format might change in future releases.
      4. If you write scripts that parse jps output, then expect to modify them for future releases of this tool.

### Example

* + - 1. List the instrumented JVMs on the local host:

> jps

18027 Java2Demo.JAR

18032 jps

18005 jstat

* + - 1. The following example lists the instrumented JVMs on a remote host.
         1. This example assumes that the jstat server and either the its internal RMI registry or a separate external rmiregistry process are running on the remote host on the default port (port 1099).
         2. It also assumes that the local host has appropriate permissions to access the remote host.
         3. This example includes the -l option to output the long form of the class names or JAR file names.

> jps -l remote.domain

3002 /opt/jdk1.8.0/demo/jfc/Java2D/Java2Demo.JAR

2857 sun.tools.jstatd.jstatd

* + - 1. The following example lists the instrumented JVMs on a remote host with a nondefault port for the RMI registry.
         1. This example assumes that the jstatd server, with an internal RMI registry bound to port 2002, is running on the remote host.
         2. This example also uses the -m option to include the arguments passed to the main method of each of the listed Java applications.

>jps -m remote.domain:2002

3002 /opt/jdk1.7.0/demo/jfc/Java2D/Java2Demo.JAR

3102 sun.tools.jstatd.jstatd -p 2002

## jstat

### Description

* + - 1. The jstat command displays performance statistics for an instrumented Java HotSpot VM.
      2. The target JVM is identified by its virtual machine identifier, or vmid option.
      3. The jstat command supports two types of options, general options and output options.
      4. General options cause the jstat command to display simple usage and version information.
      5. Output options determine the content and format of the statistical output.
      6. All options and their functionality are subject to change or removal in future releases.

### Synopsis

jstat generalOptions

jstat outputOptions [-t] [-h n] vmid [interval [count]]

|  |  |
| --- | --- |
| generalOptions | A single general command-line option.  If you specify one of the general options, then you can't specify any other option or parameter.  -help  Displays a help message.  -options  Displays a list of static options. |
| outputOption | An option reported by the -options option.  One or more output options that consist of a single statOption, plus any of the -t, -h, and -J options.  -J options.  Passes javaOption to the Java application launcher.  For example, -J-Xms48m sets the startup memory to 48 MB  see Output Options and Output |
| -t | Displays a time-stamp column as the first column of output.  The time stamp is the time since the start time of the target JVM |
| -h n | Displays a column header every n samples (output rows), where n is a positive integer. The default value is 0, which displays the column header of the first row of data. |
| vmid | A virtual machine identifier, which is a string that indicates the target JVM.  See Virtual Machine Identifier. |
| interval | The sampling interval in the specified units, seconds (s) or milliseconds (ms).  Default units are milliseconds.  This must be a positive integer.  When specified, the jstat command produces its output at each interval. |
| count | The number of samples to display.  The default value is infinity, which causes the jstat command to display statistics until the target JVM terminates or the jstat command is terminated.  This value must be a positive integer. |

### Virtual Machine Identifier

[protocol:][//]lvmid[@hostname[:port][/servername]

The syntax of the vmid string corresponds to the syntax of a URI.

The vmid string can vary from a simple integer that represents a local JVM to a more complex construction that specifies a communications protocol, port number, and other implementation-specific values.

|  |  |
| --- | --- |
| protocol | The communications protocol.  If the protocol value is omitted and a host name isn't specified, then the default protocol is a platform-specific optimized local protocol.  If the protocol value is omitted and a host name is specified, then the default protocol is rmi. |
| lvmid | The local virtual machine identifier for the target JVM.  The lvmid is a platform-specific value that uniquely identifies a JVM on a system.  The lvmid is the only required component of a virtual machine identifier.  The lvmid is typically, but not necessarily, the operating system's process identifier for the target JVM process.  You can use the jps command to determine the lvmid provided the JVM processes is not running in a separate docker instance.  You can also determine the lvmid on Linux and OS X platforms with the ps command, and on Windows with the Windows Task Manager. |
| hostname | A host name or IP address that indicates the target host.  If the hostname value is omitted, then the target host is the local host. |
| port | The default port for communicating with the remote server.  If the hostname value is omitted or the protocol value specifies an optimized, local protocol, then the port value is ignored.  Otherwise, treatment of the port parameter is implementation-specific.  For the default rmi protocol, the port value indicates the port number for the rmiregistry on the remote host.  If the port value is omitted and the protocol value indicates rmi, then the default rmiregistry port (1099) is used. |
| servername | The treatment of the servername parameter depends on implementation.  For the optimized local protocol, this field is ignored.  For the rmi protocol, it represents the name of the RMI remote object on the remote host. |

### Output Options and Output

* + - 1. If you don't specify a general option, then you can specify output options.
      2. Output options determine the content and format of the jstat command's output, and consist of a single statOption, plus any of the other output options (-h, -t, and -J).
      3. The statOption must come first.
      4. Output is formatted as a table, with columns that are separated by spaces.
      5. A header row with titles describes the columns.
      6. Use the -h option to set the frequency at which the header is displayed.
      7. Column header names are consistent among the different options.
      8. In general, if two options provide a column with the same name, then the data source for the two columns is the same.
      9. Use the -t option to display a time-stamp column, labeled Timestamp as the first column of output.
         1. The Timestamp column contains the elapsed time, in seconds, since the target JVM started.
         2. The resolution of the time stamp is dependent on various factors and is subject to variation due to delayed thread scheduling on heavily loaded systems.
         3. Use the interval and count parameters to determine how frequently and how many times, respectively, the jstat command displays its output.
      10. Note:
          1. Don't write scripts to parse the jstat command's output because the format might change in future releases.
          2. If you write scripts that parse the jstat command output, then expect to modify them for future releases of this tool.

|  |  |
| --- | --- |
| statOptions | Output |
| -class option | Displays statistics about the behavior of the class loader. |
| Loaded:  Number of classes loaded.  Bytes:  Number of KB loaded.  Unloaded:  Number of classes unloaded.  Bytes:  Number of KB loaded.  Time:  Time spent performing class loading and unloading operations. |
| -compiler option | Displays statistics about the behavior of the Java HotSpot VM Just-in-Time compiler. |
| Compiled:  Number of compilation tasks performed.  Failed:  Number of compilations tasks failed.  Invalid:  Number of compilation tasks that were invalidated.  Time:  Time spent performing compilation tasks.  FailedType:  Compile type of the last failed compilation.  FailedMethod:  Class name and method of the last failed compilation. |
| -gc option | Displays statistics about the behavior of the garbage collected heap. |
| S0C:  Current survivor space 0 capacity (KB).  S1C:  Current survivor space 1 capacity (KB).  S0U:  Survivor space 0 utilization (KB).  S1U:  Survivor space 1 utilization (KB).  EC:  Current eden space capacity (KB).  EU:  Eden space utilization (KB).  OC:  Current old space capacity (KB).  OU:  Old space utilization (KB).  MC:  Metaspace Committed Size (KB).  MU:  Metaspace utilization (KB).  CCSC:  Compressed class committed size (KB).  CCSU:  Compressed class space used (KB).  YGC:  Number of young generation garbage collection (GC) events.  YGCT:  Young generation garbage collection time.  FGC:  Number of full GC events.  FGCT:  Full garbage collection time.  GCT:  Total garbage collection time. |
| -gccapacity option | Displays statistics about the capacities of the generations and their corresponding spaces. |
| NGCMN:  Minimum new generation capacity (KB).  NGCMX:  Maximum new generation capacity (KB).  NGC:  Current new generation capacity (KB).  S0C:  Current survivor space 0 capacity (KB).  S1C:  Current survivor space 1 capacity (KB).  EC:  Current eden space capacity (KB).  OGCMN:  Minimum old generation capacity (KB).  OGCMX:  Maximum old generation capacity (KB).  OGC:  Current old generation capacity (KB).  OC:  Current old space capacity (KB).  MCMN:  Minimum metaspace capacity (KB).  MCMX:  Maximum metaspace capacity (KB).  MC:  Metaspace Committed Size (KB).  CCSMN:  Compressed class space minimum capacity (KB).  CCSMX:  Compressed class space maximum capacity (KB).  CCSC:  Compressed class committed size (KB).  YGC:  Number of young generation GC events.  FGC:  Number of full GC events. |
| -gccause option | Displays a summary about garbage collection statistics with the cause of the last and current (when applicable) garbage collection events.  This option displays the same summary of garbage collection statistics as the -gcutil option, but includes the causes of the last garbage collection event and (when applicable), the current garbage collection event.  In addition to the columns listed for -gcutil, this option adds the following columns: |
| LGCC:  Cause of last garbage collection  GCC:  Cause of current garbage collection |
| -gcnew option | Displays statistics about the behavior of the new generation. |
| S0C:  Current survivor space 0 capacity (KB).  S1C:  Current survivor space 1 capacity (KB).  S0U:  Survivor space 0 utilization (KB).  S1U:  Survivor space 1 utilization (KB).  TT:  Tenuring threshold.  MTT:  Maximum tenuring threshold.  DSS:  Desired survivor size (KB).  EC:  Current eden space capacity (KB).  EU:  Eden space utilization (KB).  YGC:  Number of young generation GC events.  YGCT:  Young generation garbage collection time. |
| -gcnewcapacity option | Displays statistics about the sizes of the new generations and their corresponding spaces. |
| NGCMN:  Minimum new generation capacity (KB).  NGCMX:  Maximum new generation capacity (KB).  NGC:  Current new generation capacity (KB).  S0CMX:  Maximum survivor space 0 capacity (KB).  S0C:  Current survivor space 0 capacity (KB).  S1CMX:  Maximum survivor space 1 capacity (KB).  S1C:  Current survivor space 1 capacity (KB).  ECMX:  Maximum eden space capacity (KB).  EC:  Current eden space capacity (KB).  YGC:  Number of young generation GC events.  FGC:  Number of full GC events. |
| -gcold option | Displays statistics about the behavior of the old generation and metaspace statistics. |
| MC:  Metaspace Committed Size (KB).  MU:  Metaspace utilization (KB).  CCSC:  Compressed class committed size (KB).  CCSU:  Compressed class space used (KB).  OC:  Current old space capacity (KB).  OU:  Old space utilization (KB).  YGC:  Number of young generation GC events.  FGC:  Number of full GC events.  FGCT:  Full garbage collection time.  GCT:  Total garbage collection time. |
| -gcoldcapacity option | Displays statistics about the sizes of the old generation. |
| OGCMN:  Minimum old generation capacity (KB).  OGCMX:  Maximum old generation capacity (KB).  OGC:  Current old generation capacity (KB).  OC:  Current old space capacity (KB).  YGC:  Number of young generation GC events.  FGC:  Number of full GC events.  FGCT:  Full garbage collection time.  GCT:  Total garbage collection time. |
| -gcmetacapacity option | Displays statistics about the sizes of the metaspace. |
| MCMN:  Minimum metaspace capacity (KB).  MCMX:  Maximum metaspace capacity (KB).  MC:  Metaspace Committed Size (KB).  CCSMN:  Compressed class space minimum capacity (KB).  CCSMX:  Compressed class space maximum capacity (KB).  YGC:  Number of young generation GC events.  FGC:  Number of full GC events.  FGCT:  Full garbage collection time.  GCT:  Total garbage collection time. |
| -gcutil option | Displays a summary about garbage collection statistics. |
| GC简要统计  S0:  Survivor space 0 utilization as a percentage of the space's current capacity.  S1:  Survivor space 1 utilization as a percentage of the space's current capacity.  E:  Eden space utilization as a percentage of the space's current capacity.  O:  Old space utilization as a percentage of the space's current capacity.  M:  Metaspace utilization as a percentage of the space's current capacity.  CCS:  Compressed class space utilization as a percentage.  YGC:  Number of young generation GC events.  YGCT:  Young generation garbage collection time.  FGC:  Number of full GC events.  FGCT:  Full garbage collection time.  GCT:  Total garbage collection time. |
| -printcompilation option | Displays Java HotSpot VM compilation method statistics. |
| Compiled:  Number of compilation tasks performed by the most recently compiled method.  Size:  Number of bytes of byte code of the most recently compiled method.  Type:  Compilation type of the most recently compiled method.  Method:  Class name and method name identifying the most recently compiled method.  Class name uses a slash (/) instead of a dot (.) as a name space separator.  The method name is the method within the specified class.  The format for these two fields is consistent with the HotSpot -XX:+PrintCompilation option. |

### Example

* + - 1. This example attaches to lvmid 21891 and takes 7 samples at 250 millisecond intervals and displays the output as specified by the -gcutil option.

The output of this example shows that a young generation collection occurred between the third and fourth sample. The collection took 0.078 seconds and promoted objects from the eden space (E) to the old space (O), resulting in an increase of old space utilization from 66.80% to 68.19%. Before the collection, the survivor space was 97.02% utilized, but after this collection it's 91.03% utilized.

>jstat -gcutil 21891 250 7

S0 S1 E O M CCS YGC YGCT FGC FGCT GCT

0.00 97.02 70.31 66.80 95.52 89.14 7 0.300 0 0.000 0.300

0.00 97.02 86.23 66.80 95.52 89.14 7 0.300 0 0.000 0.300

0.00 97.02 96.53 66.80 95.52 89.14 7 0.300 0 0.000 0.300

91.03 0.00 1.98 68.19 95.89 91.24 8 0.378 0 0.000 0.378

91.03 0.00 15.82 68.19 95.89 91.24 8 0.378 0 0.000 0.378

91.03 0.00 17.80 68.19 95.89 91.24 8 0.378 0 0.000 0.378

91.03 0.00 17.80 68.19 95.89 91.24 8 0.378 0 0.000 0.378

* + - 1. Repeat the Column Header String

This example attaches to lvmid 21891 and takes samples at 250 millisecond intervals and displays the output as specified by -gcnew option. In addition, it uses the -h3 option to output the column header after every 3 lines of data.

In addition to showing the repeating header string, this example shows that between the second and third samples, a young GC occurred. Its duration was 0.001 seconds. The collection found enough active data that the survivor space 0 utilization (S0U) would have exceeded the desired survivor size (DSS). As a result, objects were promoted to the old generation (not visible in this output), and the tenuring threshold (TT) was lowered from 31 to 2.

Another collection occurs between the fifth and sixth samples. This collection found very few survivors and returned the tenuring threshold to 31.

>jstat -gcnew -h3 21891 250

S0C S1C S0U S1U TT MTT DSS EC EU YGC YGCT

64.0 64.0 0.0 31.7 31 31 32.0 512.0 178.6 249 0.203

64.0 64.0 0.0 31.7 31 31 32.0 512.0 355.5 249 0.203

64.0 64.0 35.4 0.0 2 31 32.0 512.0 21.9 250 0.204

S0C S1C S0U S1U TT MTT DSS EC EU YGC YGCT

64.0 64.0 35.4 0.0 2 31 32.0 512.0 245.9 250 0.204

64.0 64.0 35.4 0.0 2 31 32.0 512.0 421.1 250 0.204

64.0 64.0 0.0 19.0 31 31 32.0 512.0 84.4 251 0.204

S0C S1C S0U S1U TT MTT DSS EC EU YGC YGCT

64.0 64.0 0.0 19.0 31 31 32.0 512.0 306.7 251 0.204

* + - 1. Include a Time Stamp for Each Sample

This example attaches to lvmid 21891 and takes 3 samples at 250 millisecond intervals. The -t option is used to generate a time stamp for each sample in the first column.

The Timestamp column reports the elapsed time in seconds since the start of the target JVM. In addition, the -gcoldcapacity output shows the old generation capacity (OGC) and the old space capacity (OC) increasing as the heap expands to meet allocation or promotion demands. The old generation capacity (OGC) has grown from 11,696 KB to 13,820 KB after the eighty-first full garbage collection (FGC). The maximum capacity of the generation (and space) is 60,544 KB (OGCMX), so it still has room to expand.

Timestamp OGCMN OGCMX OGC OC YGC FGC FGCT GCT

150.1 1408.0 60544.0 11696.0 11696.0 194 80 2.874 3.799

150.4 1408.0 60544.0 13820.0 13820.0 194 81 2.938 3.863

150.7 1408.0 60544.0 13820.0 13820.0 194 81 2.938 3.863

* + - 1. Monitor Instrumentation for a Remote JVM

This example attaches to lvmid 40496 on the system named remote.domain using the -gcutil option, with samples taken every second indefinitely.

The lvmid is combined with the name of the remote host to construct a vmid of 40496@remote.domain. This vmid results in the use of the rmi protocol to communicate to the default jstatd server on the remote host. The jstatd server is located using the rmiregistry command on remote.domain that's bound to the default port of the rmiregistry command (port 1099).

>jstat -gcutil 40496@remote.domain 1000

... output omitted

## jinfo

### Description

* + - 1. The jinfo command prints Java configuration information for a specified Java process.
      2. The configuration information includes Java system properties and JVM command-line flags.
      3. If the specified process is running on a 64-bit JVM, then you might need to specify the -J-d64 option,

for example:

jinfo -J-d64 -sysprops *pid*

* + - 1. This command is unsupported and might not be available in future releases of the JDK.
      2. In Windows Systems
         1. If dbgeng.dll is not present, the Debugging Tools for Windows must be installed to have these tools work.
         2. The PATH environment variable should contain the location of the jvm.dll that's used by the target process or the location from which the core dump file was produced.

### Synopsis

jinfo [option] pid

|  |  |
| --- | --- |
| pid | The process ID for which the configuration information is to be printed.  The process must be a Java process.  To get a list of Java processes running on a machine, use either the ps command or, if the JVM processes are not running in a separate docker instance, the jps command. |
| option | If none of the following options are used, both the command-line flags and the system property name-value pairs are printed.  -flag name  Prints the name and value of the specified command-line flag.  -flag [+|-]name  Enables or disables the specified Boolean command-line flag.  -flag name=value  Sets the specified command-line flag to the specified value.  -flags  Prints command-line flags passed to the JVM.  -sysprops  Prints Java system properties as name-value pairs.  -h or -help  Prints a help message. |

## jmap

### Description

* + - 1. The jmap command prints details of a specified running process.
      2. Note:
         1. This command is unsupported and might not be available in future releases of the JDK.
         2. On Windows Systems where the dbgeng.dll file isn't present, the Debugging Tools for Windows must be installed to make these tools work.
         3. The PATH environment variable should contain the location of the jvm.dll file that's used by the target process or the location from which the core dump file was produced.

### Synopsis

jmap [options] pid

|  |  |
| --- | --- |
| pid | The process ID for which the information specified by the options is to be printed. The process must be a Java process.  To get a list of Java processes running on a machine, use either the ps command or, if the JVM processes are not running in a separate docker instance, the jps command. |
| Options | This represents the jmap command-line options.  -clstats pid  Connects to a running process and prints class loader statistics of Java heap.  -finalizerinfo pid  Connects to a running process and prints information on objects awaiting finalization.  -histo[:live] pid  Connects to a running process and prints a histogram of the Java object heap.  If the live suboption is specified, it then counts only live objects.  -dump:dump\_options pid  Connects to a running process and dumps the Java heap. The dump\_options include:  live  When specified, dumps only the live objects;  if not specified, then dumps all objects in the heap.  format=b  Dumps the Java heap in hprof binary format  file=filename  Dumps the heap to filename |

### Example

jmap -dump:live,format=b,file=heap.bin *pid*

## jhat

在jdk9+被移除，官方建议用visualVm代替

## jstack

### Description

* + - 1. This command is experimental and unsupported.
      2. The jstack command prints Java stack traces of Java threads for a specified Java process.
      3. For each Java frame, the full class name, method name, byte code index (BCI), and line number, when available, are printed.
      4. C++ mangled names aren't demangled.
      5. To demangle C++ names, the output of this command can be piped to c++filt.
      6. When the specified process is running on a 64-bit JVM, you might need to specify the -J-d64 option,

for example: jstack -J-d64 pid.

### Synopsis

jstack [options] pid

|  |  |
| --- | --- |
| options | This represents the jstack command-line options.  Option：  -l  The long listing option prints additional information about locks.  -h or -help  Prints a help message. |
| pid | The process ID for which the stack trace is printed.  The process must be a Java process.  To get a list of Java processes running on a machine, use either the ps command or, if the JVM processes are not running in a separate docker instance, the jps command. |

## jstatd

### Description

* + - 1. This command is experimental and unsupported.
      2. The jstatd command is an RMI server application that monitors for the creation and termination of instrumented Java HotSpot VMs and provides an interface to enable remote monitoring tools, jstat and jps, to attach to JVMs that are running on the local host and collect information about the JVM process.
      3. The jstatd server requires an RMI registry on the local host.
      4. The jstatd server attempts to attach to the RMI registry on the default port, or on the port you specify with the -p port option.
      5. If an RMI registry is not found, then one is created within the jstatd application that's bound to the port that's indicated by the -p port option or to the default RMI registry port when the -p port option is omitted.
      6. You can stop the creation of an internal RMI registry by specifying the -nr option.
      7. Remote Interface
         1. The interface exported by the jstatd process is proprietary and guaranteed to change.
         2. Users and developers are discouraged from writing to this interface.

### Synopsis

jstatd [options]

Options

This represents the jstatd command-line options. See Options for the jstatd Command.

-nr

This option does not attempt to create an internal RMI registry within the jstatd process when an existing RMI registry isn't found.

-p port

This option sets the port number where the RMI registry is expected to be found, or when not found, created if the -nr option isn't specified.

-r rmiport

This option sets the port number to which the RMI connector is bound.

If not specified a random available port is used.

-n rminame

This option sets the name to which the remote RMI object is bound in the RMI registry.

The default name is JStatRemoteHost.

If multiple jstatd servers are started on the same host, then the name of the exported RMI object for each server can be made unique by specifying this option.

However, doing so requires that the unique server name be included in the monitoring client's hostid and vmid strings.

-Joption

This option passes a Java option to the JVM, where the option is one of those described on the reference page for the Java application launcher.

For example, -J-Xms48m sets the startup memory to 48 MB. See java.

### Security

* + - 1. The jstatd server can monitor only JVMs for which it has the appropriate native access permissions.
      2. Therefore, the jstatd process must be running with the same user credentials as the target JVMs.
      3. Some user credentials, such as the root user in Linux and OS X operating systems, have permission to access the instrumentation exported by any JVM on the system.
      4. A jstatd process running with such credentials can monitor any JVM on the system, but introduces additional security concerns.
      5. The jstatd server doesn't provide any authentication of remote clients.
      6. Therefore, running a jstatd server process exposes the instrumentation export by all JVMs for which the jstatd process has access permissions to any user on the network.
      7. This exposure might be undesirable in your environment, and therefore, local security policies should be considered before you start the jstatd process, particularly in production environments or on networks that aren't secure.
      8. The jstatd server installs an instance of RMISecurityPolicy when no other security manager is installed, and therefore, requires a security policy file to be specified.
      9. The policy file must conform to Default Policy Implementation and Policy File Syntax.
      10. If your security concerns can't be addressed with a customized policy file, then the safest action is to not run the jstatd server and use the jstat and jps tools locally.
      11. However, when using jps to get a list of instrumented JVMs, the list will not include any JVMs running in docker containers.

### Example

The following are examples of the jstatd command. The jstatd scripts automatically start the server in the background.

* + - 1. Internal RMI Registry

This example shows how to start a jstatd session with an internal RMI registry. This example assumes that no other server is bound to the default RMI registry port (port 1099).

jstatd -J-Djava.security.policy=all.policy

* + - 1. External RMI Registry
         1. This example starts a jstatd session with an external RMI registry.

rmiregistry&

jstatd -J-Djava.security.policy=all.policy

* + - * 1. This example starts a jstatd session with an external RMI registry server on port 2020.

jrmiregistry 2020&

jstatd -J-Djava.security.policy=all.policy -p 2020

* + - * 1. This example starts a jstatd session with an external RMI registry server on port 2020 and JMX connector bound to port 2021.

jrmiregistry 2020&

jstatd -J-Djava.security.policy=all.policy -p 2020 -r 2021

* + - * 1. This example starts a jstatd session with an external RMI registry on port 2020 that's bound to AlternateJstatdServerName.

rmiregistry 2020&

jstatd -J-Djava.security.policy=all.policy -p 2020 -n AlternateJstatdServerName

* + - 1. Stop the Creation of an In-Process RMI Registry

This example starts a jstatd session that doesn't create an RMI registry when one isn't found. This example assumes an RMI registry is already running. If an RMI registry isn't running, then an error message is displayed.

jstatd -J-Djava.security.policy=all.policy -nr

* + - 1. Enable RMI Logging

This example starts a jstatd session with RMI logging capabilities enabled. This technique is useful as a troubleshooting aid or for monitoring server activities.

jstatd -J-Djava.security.policy=all.policy -J-Djava.rmi.server.logCalls=true

## jcmd

### Description

* + - 1. The jcmd utility is used to send diagnostic command requests to the JVM.
      2. It must be used on the same machine on which the JVM is running, and have the same effective user and group identifiers that were used to launch the JVM.
      3. Each diagnostic command has its own set of arguments.
      4. To display the description, syntax, and a list of available arguments for a diagnostic command, use the name of the command as the argument. For example:

jcmd pid help command

* + - 1. If arguments contain spaces, then you must surround them with single or double quotation marks (' or ").
      2. In addition, you must escape single or double quotation marks with a backslash (\) to prevent the operating system shell from processing quotation marks.
      3. Alternatively, you can surround these arguments with single quotation marks and then with double quotation marks (or with double quotation marks and then with single quotation marks).
      4. If you specify the process identifier (pid) or the main class (main-class) as the first argument, then the jcmd utility sends the diagnostic command request to the Java process with the specified identifier or to all Java processes with the specified name of the main class. You can also send the diagnostic command request to all available Java processes by specifying 0 as the process identifier.

### Synopsis

jcmd [pid | main-class] command... | PerfCounter.print | -f filename

jcmd [-l]

jcmd -h

|  |  |
| --- | --- |
| pid | When used, the jcmd utility sends the diagnostic command request to the process ID for the Java process. |
| main-class | When used, the jcmd utility sends the diagnostic command request to all Java processes with the specified name of the main class. |
| command | 1. The command must be a valid jcmd command for the selected JVM. 2. The list of available commands for jcmd is obtained by running the help command (jcmd pid help) where pid is the process ID for the running Java process. 3. If the pid is 0, commands will be sent to **all** Java processes. 4. The main class argument will be used to match, either partially or fully, the class used to start Java. 5. If no options are given, it lists the running Java process identifiers with the main class and command-line arguments that were used to launch the process (the same as using -l). |
| Perfcounter.print | Prints the performance counters exposed by the specified Java process. |
| -f filename | Reads and executes commands from a specified file, filename. |
| -l | 1. Displays the list of Java Virtual Machine process identifiers that are not running in a separate docker process along with the main class and command-line arguments that were used to launch the process. 2. If the JVM is in a docker process, you must use tools such as ps to look up the PID. 3. Note: Using jcmd without arguments is the same as using jcmd -l. |
| -h | Displays the jcmd utility's command-line help. |

### Commands for jcmd

|  |  |
| --- | --- |
| help [options] [arguments] | For more information about a specific command.  arguments:  command name: The name of the command for which we want help (STRING, no default value)  Note:The following options must be specified using either key or key=value syntax.  options:  -all: (Optional) Show help for all commands (BOOLEAN, false) . |
| Compiler.codecache | Prints code cache layout and bounds.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| Compiler.codelist | Prints all compiled methods in code cache that are alive.  Impact: Medium  Permission: java.lang.management.ManagementPermission(monitor) |
| Compiler.queue | Prints methods queued for compilation.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| Compiler.directives\_add \*filename\* \*arguments\* | Adds compiler directives from a file.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor)  arguments:  filename: The name of the directives file (STRING, no default value) |
| Compiler.directives\_clear | Remove all compiler directives.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| Compiler.directives\_print | Prints all active compiler directives.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| Compiler.directives\_remove | Remove latest added compiler directive.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| GC.class\_histogram [options] | Provides statistics about the Java heap usage.  Impact: High --- depends on Java heap size and content.  Permission: java.lang.management.ManagementPermission(monitor)  Note:The options must be specified using either key or key=value syntax.  options:  -all: (Optional) Inspects all objects, including unreachable objects (BOOLEAN, false) |
| GC.finalizer\_info | Provides information about the Java finalization queue.  Impact: Medium  Permission: java.lang.management.ManagementPermission(monitor) |
| GC.heap\_dump [options] [arguments] | Generates a HPROF format dump of the Java heap.  Impact: High --- depends on the Java heap size and content. Request a full GC unless the -all option is specified.  Permission: java.lang.management.ManagementPermission(monitor)  Note:The following options must be specified using either key or key=value syntax.  options:  -all: (Optional) Dump all objects, including unreachable objects (BOOLEAN, false)  arguments:  filename: The name of the dump file (STRING, no default value) |
| GC.heap\_info | Provides generic Java heap information.  Impact: Medium  Permission: java.lang.management.ManagementPermission(monitor) |
| GC.run | Calls java.lang.System.gc().  Impact: Medium --- depends on the Java heap size and content. |
| GC.run\_finalization | Calls java.lang.System.runFinalization().  Impact: Medium --- depends on the Java content. |
| JFR.check [options] | Show information about a running flight recording  Impact: Low  Note:The options must be specified using either key or key=value syntax. If no parameters are entered, information for all active recordings is shown.  options:  name: (Optional) Name of the flight recording. (STRING, no default value)  verbose: (Optional) Flag for printing the event settings for the recording (BOOLEAN, false) |
| JFR.configure [options] | Set the parameters for a flight recording  Impact: Low  Note:The options must be specified using either key or key=value syntax. If no parameters are entered, the current settings are displayed.  options:  globalbuffercount:  (Optional) Number of global buffers. This option is a legacy option: change the memorysize parameter to alter the number of global buffers. This value cannot be changed once JFR has been initalized. (STRING, default determined by the value for memorysize)  globalbuffersize:  (Optional) Size of the global buffers, in bytes. This option is a legacy option: change the memorysize parameter to alter the size of the global buffers. This value cannot be changed once JFR has been initalized. (STRING, default determined by the value for memorysize)  maxchunksize:  (Optional) Maximum size of an individual data chunk in bytes if one of the following suffixes is not used: 'm' or 'M' for megabytes OR 'g' or 'G' for gigabytes. This value cannot be changed once JFR has been initialized. (STRING, 12M)  memorysize:  (Optional) Overall memory size, in bytes if one of the following suffixes is not used: 'm' or 'M' for megabytes OR 'g' or 'G' for gigabytes. This value cannot be changed once JFR has been initialized. (STRING, 10M)  repositorypath:  (Optional) Path to the location where recordings are stored until they are written to a permanent file. (STRING, The default location is the temporary directory for the operating system. On Linux operating systems, the temporary directory is /tmp. On Windwows, the temporary directory is specified by the TMP environment variable.)  stackdepth:  (Optional) Stack depth for stack traces. Setting this value greater than the default of 64 may cause a performance degradation. This value cannot be changed once JFR has been initialized. (LONG, 64)  thread\_buffer\_size:  (Optional) Local buffer size for each thread in bytes if one of the following suffixes is not used: 'k' or 'K' for kilobytes or 'm' or 'M' for megabytes. Overriding this parameter could reduce performance and is not recommended. This value cannot be changed once JFR has been initialized. (STRING, 8k)  samplethreads:  (Optional) Flag for activating thread sampling. (BOOLEAN, true) |
| JFR.dump [options] | Write data to a file while a flight recording is running  Impact: Low  Note:The options must be specified using either key or key=value syntax.  No options are required.  The recording continues to run after the data is written.  options:  begin:  (Optional) Specify the time from which recording data will be included in the dump file.  The format is specified as local time.  (STRING, no default value)  end:  (Optional) Specify the time to which recording data will be included in the dump file.  The format is specified as local time.  (STRING, no default value)  Note:  For both begin and end, the time must be in a format that can be read by java.time.LocalTime::parse(STRING), java.time.LocalDateTime::parse(STRING) or java.time.Instant::parse(STRING).  For example, "13:20:15", "2020-03-17T09:00:00" or "2020-03-17T09:00:00Z".  Note:  begin and end times correspond to the timestamps found within the recorded information in the flight recording data.  Another option is to use a time relative to the current time that is specified by a negative integer followed by "s", "m" or "h". For example, "-12h", "-15m" or "-30s"  filename:  (Optional) Name of the file to which the flight recording data is dumped.  If no filename is given, a filename is generated from the PID and the current date.  The filename may also be a directory in which case, the filename is generated from the PID and the current date in the specified directory.  (STRING, no default value)  maxage:  (Optional) Length of time for dumping the flight recording data to a file.  (INTEGER followed by 's' for seconds 'm' for minutes or 'h' for hours, no default value)  maxsize:  (Optional) Maximum size for the amount of data to dump from a flight recording in bytes if one of the following suffixes is not used: 'm' or 'M' for megabytes OR 'g' or 'G' for gigabytes. (STRING, no default value)  name:  (Optional) Name of the recording.  If no name is given, data from all recordings is dumped. (STRING, no default value)  path-to-gc-root:  (Optional) Flag for saving the path to garbage collection (GC) roots at the time the recording data is dumped.  The path information is useful for finding memory leaks but collecting it can cause the application to pause for a short period of time.  Turn on this flag only when you have an application that you suspect has a memory leak. (BOOLEAN, false) |
| JFR.start [options] | Start a flight recording  Impact: Low  Note:  The options must be specified using either key or key=value syntax. If no parameters are entered, then a recording is started with default values.  options:  delay:  (Optional) Length of time to wait before starting to record (INTEGER followed by 's' for seconds 'm' for minutes or 'h' for hours, 0s)  disk:  (Optional) Flag for also writing the data to disk while recording (BOOLEAN, true)  dumponexit:  (Optional) Flag for writing the recording to disk when the Java Virtual Machine (JVM) shuts down.  If set to 'true' and no value is given for filename, the recording is written to a file in the directory where the process was started.  The file name is a system-generated name that contains the process ID, the recording ID and the current time stamp.  (For example: id-1-2019\_12\_12\_10\_41.jfr)  (BOOLEAN, false)  duration:  (Optional) Length of time to record.  Note that 0s means forever  (INTEGER followed by 's' for seconds 'm' for minutes or 'h' for hours, 0s)  filename:  (Optional) Name of the file to which the flight recording data is written when the recording is stopped.  If no filename is given, a filename is generated from the PID and the current date and is placed in the directory where the process was started.  The filename may also be a directory in which case, the filename is generated from the PID and the current date in the specified directory.  (STRING, no default value)  maxage:  (Optional) Maximum time to keep the recorded data on disk.  This parameter is valid only when the disk parameter is set to true.  Note 0s means forever.  (INTEGER followed by 's' for seconds 'm' for minutes or 'h' for hours, 0s)  maxsize:  (Optional) Maximum size of the data to keep on disk in bytes if one of the following suffixes is not used: 'm' or 'M' for megabytes OR 'g' or 'G' for gigabytes.  This parameter is valid only when the disk parameter is set to 'true'.  The value must not be less than the value for the maxchunksize parameter set with the JFR.  configure command.  (STRING, 0 (no maximum size))  name:  (Optional) Name of the recording.  If no name is provided, a name is generated.  Make note of the generated name that is shown in the response to the command so that you can use it with other commands.  (STRING, system-generated default name)  path-to-gc-root:  (Optional) Flag for saving the path to garbage collection (GC) roots at the end of a recording.  The path information is useful for finding memory leaks but collecting it is time consuming.  Turn on this flag only when you have an application that you suspect has a memory leak.  If the settings parameter is set to 'profile', then the information collected includes the stack trace from where the potential leaking object was allocated.  (BOOLEAN, false)  settings:  (Optional) Name of the settings file that identifies which events to record.  To specify more than one file, separate the names with a comma (',').  Include the path if the file is not in JAVA-HOME/lib/jfr.  The following profiles are included with the JDK in the JAVA-HOME/lib/jfr directory: 'default.jfc': collects a predefined set of information with low overhead, so it has minimal impact on performance and can be used with recordings that run continuously; 'profile.jfc': Provides more data than the 'default.jfc' profile, but with more overhead and impact on performance.  Use this configuration for short periods of time when more information is needed.  Use none to start a recording without a predefined configuration file.  (STRING, JAVA-HOME/lib/jfr/default.jfc) |
| JFR.stop [options] | Stop a flight recording  Impact: Low  Note:  The options must be specified using either key or key=value syntax. If no parameters are entered, then no recording is stopped.  options:  filename:  (Optional) Name of the file to which the recording is written when the recording is stopped.  If no path is provided, the data from the recording is discarded.  (STRING, no default value)  name:  (Optional) Name of the recording  (STRING, no default value) |
| JVMTI.agent\_load [arguments] | Loads JVMTI native agent.  Impact: Low  Permission: java.lang.management.ManagementPermission(control)  arguments:  library path:  Absolute path of the JVMTI agent to load.  (STRING, no default value)  agent option:  (Optional) Option string to pass the agent.  (STRING, no default value) |
| JVMTI.data\_dump | Signals the JVM to do a data-dump request for JVMTI.  Impact: High  Permission: java.lang.management.ManagementPermission(monitor) |
| ManagementAgent.start [options] | Starts remote management agent.  Impact: Low --- no impact  Note:  The following options must be specified using either key or key=value syntax.  options:  config.file:  (Optional) Sets com.sun.management.config.file  (STRING, no default value)  jmxremote.host:  (Optional) Sets com.sun.management.jmxremote.host  (STRING, no default value)  jmxremote.port:  (Optional) Sets com.sun.management.jmxremote.port  (STRING, no default value)  jmxremote.rmi.port:  (Optional) Sets com.sun.management.jmxremote.rmi.port  (STRING, no default value)  jmxremote.ssl:  (Optional) Sets com.sun.management.jmxremote.ssl  (STRING, no default value)  jmxremote.registry.ssl:  (Optional)Sets com.sun.management.jmxremote.registry.ssl  (STRING, no default value)  jmxremote.authenticate:  (Optional)Sets com.sun.management.jmxremote.authenticate  (STRING, no default value)  jmxremote.password.file:  (Optional)Sets com.sun.management.jmxremote.password.file  (STRING, no default value)  jmxremote.access.file:  (Optional) Sets com.sun.management.jmxremote.acce ss.file  (STRING, no default value)  jmxremote.login.config:  (Optional) Sets com.sun.management.jmxremote.log in.config  (STRING, no default value)  jmxremote.ssl.enabled.cipher.suites:  (Optional) Sets com.sun.management.  jmxremote.ssl.enabled.cipher.suite:  (STRING, no default value)  jmxremote.ssl.enabled.protocols:  (Optional)Sets com.sun.management.jmxr emote.ssl.enabled.protocols  (STRING, no default value)  jmxremote.ssl.need.client.auth:  (Optional)Sets com.sun.management.jmxre mote.need.client.auth  (STRING, no default value)  jmxremote.ssl.config.file:  (Optional) Sets com.sun.management.jmxremote. ssl\_config\_file  (STRING, no default value)  jmxremote.autodiscovery:  (Optional)Sets com.sun.management.jmxremote.au todiscovery  (STRING, no default value)  jdp.port:  (Optional) Sets com.sun.management.jdp.port  (INT, no default value)  jdp.address:  (Optional) Sets com.sun.management.jdp.address  (STRING, no default value)  jdp.source\_addr:  (Optional) Sets com.sun.management.jdp.source\_addr  (STRING, no default value)  jdp.ttl:  (Optional) Sets com.sun.management.jdp.ttl  (INT, no default value)  jdp.pause:  (Optional) Sets com.sun.management.jdp.pause  (INT, no default value)  jdp.name:  (Optional) Sets com.sun.management.jdp.name  (STRING, no default value) |
| ManagementAgent.start\_local | Starts the local management agent.  Impact: Low --- no impact |
| ManagementAgent.status | Print the management agent status.  Impact: Low --- no impact  Permission: java.lang.management.ManagementPermission(monitor) |
| ManagementAgent.stop | Stops the remote management agent.  Impact: Low --- no impact |
| Thread.print [options] | Prints all threads with stacktraces.  Impact: Medium --- depends on the number of threads.  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax.  options:  -l:  (Optional) Prints java.util.concurrent locks  (BOOLEAN, false) |
| VM.classloader\_stats | Prints statistics about all ClassLoaders.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| VM.class\_hierarchy [options] [arguments] | Prints a list of all loaded classes, indented to show the class hierarchy.  The name of each class is followed by the ClassLoaderData\* of its ClassLoader, or "null" if it is loaded by the bootstrap class loader.  Impact: Medium --- depends on the number of loaded classes.  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax.  options:  -i:  (Optional) Inherited interfaces should be printed.  (BOOLEAN, false)  -s:  (Optional) If a class name is specified, it prints the subclasses.  If the class name is not specified, only the superclasses are printed. (BOOLEAN, false)  arguments:  classname:  (Optional) The name of the class whose hierarchy should be printed.  If not specified, all class hierarchies are printed.  (STRING, no default value) |
| VM.command\_line | Prints the command line used to start this VM instance.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| VM.dynlibs | Prints the loaded dynamic libraries.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| VM.info | Prints information about the JVM environment and status.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor) |
| VM.log [options] | Lists current log configuration, enables/disables/configures a log output, or rotates all logs.  Impact: Low  Permission: java.lang.management.ManagementPermission(control)  Note:  The following options must be specified using either key or key=value syntax.  options:  output:  (Optional) The name or index (#) of output to configure.  (STRING, no default value)  output\_options:  (Optional) Options for the output.  (STRING, no default value)  what:  (Optional) Configures what tags to log.  (STRING, no default value )  decorators:  (Optional) Configures which decorators to use.  Use 'none' or an empty value to remove all.  (STRING, no default value)  disable:  (Optional) Turns off all logging and clears the log configuration.  (BOOLEAN, no default value)  list:  (Optional) Lists current log configuration.  (BOOLEAN, no default value)  rotate:  (Optional) Rotates all logs.  (BOOLEAN, no default value) |
| VM.flags [options] | Prints the VM flag options and their current values.  Impact: Low  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax.  options:  -all:  (Optional) Prints all flags supported by the VM  (BOOLEAN, false). |
| VM.native\_memory [options] | Prints native memory usage  Impact: Medium  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax.  options:  summary:  (Optional) Requests runtime to report current memory summary, which includes total reserved and committed memory, along with memory usage summary by each subsystem.  (BOOLEAN, false)  detail:  (Optional) Requests runtime to report memory allocation >= 1K by each callsite.  (BOOLEAN, false)  baseline:  (Optional) Requests runtime to baseline current memory usage, so it can be compared against in later time.  (BOOLEAN, false)  summary.diff:  (Optional) Requests runtime to report memory summary comparison against previous baseline.  (BOOLEAN, false)  detail.diff:  (Optional) Requests runtime to report memory detail comparison against previous baseline, which shows the memory allocation activities at different callsites.  (BOOLEAN, false)  shutdown:  (Optional) Requests runtime to shutdown itself and free the memory used by runtime.  (BOOLEAN, false)  statistics:  (Optional) Prints tracker statistics for tuning purpose.  (BOOLEAN, false)  scale:  (Optional) Memory usage in which scale, KB, MB or GB  (STRING, KB) |
| VM.print\_touched\_methods | Prints all methods that have ever been touched during the lifetime of this JVM.  Impact: Medium --- depends on Java content. |
| VM.set\_flag [arguments] | Sets the VM flag option by using the provided value.  Impact: Low  Permission: java.lang.management.ManagementPermission(control)  arguments:  flag name:  The name of the flag that you want to set  (STRING, no default value)  string value:  (Optional) The value that you want to set  (STRING, no default value) |
| VM.stringtable [options] | Dumps the string table.  Impact: Medium --- depends on the Java content.  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax.  options:  -verbose:  (Optional) Dumps the content of each string in the table  (BOOLEAN, false) |
| VM.symboltable [options] | Dumps the symbol table.  Impact: Medium --- depends on the Java content.  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax).  options:  -verbose: (Optional) Dumps the content of each symbol in the table (BOOLEAN, false) |
| VM.systemdictionary | Prints the statistics for dictionary hashtable sizes and bucket length.  Impact: Medium  Permission: java.lang.management.ManagementPermission(monitor)  Note:  The following options must be specified using either key or key=value syntax.  options:  verbose: (Optional) Dump the content of each dictionary entry for all class loaders (BOOLEAN, false) . |
| VM.system\_properties | Prints the system properties.  Impact: Low  Permission: java.util.PropertyPermission(\*, read) |
| VM.uptime [options] | Prints the VM uptime.  Impact: Low  Note:  The following options must be specified using either key or key=value syntax.  options:  -date: (Optional) Adds a prefix with the current date (BOOLEAN, false) |
| VM.version | Prints JVM version information.  Impact: Low  Permission: java.util.PropertyPermission(java.vm.version, read) |

# Frequently Execution Instruction

## java

### Description

* + - 1. The java command starts a Java application.
      2. It does this by starting the Java Virtual Machine (JVM), loading the specified class, and calling that class's main() method.
         1. The method must be declared public and static,
         2. it must not return any value,
         3. it must accept a String array as a parameter.
         4. The method declaration has the following form:

public static void main(String[] args)

* + - 1. In source-file mode, the java command can launch a class declared in a source file. See Using Source-File Mode to Launch Single-File Source-Code Programs for a description of using the source-file mode.
      2. Note: You can use the JDK\_JAVA\_OPTIONS launcher environment variable to prepend its content to the actual command line of the java launcher.See Using the JDK\_JAVA\_OPTIONS Launcher Environment Variable.
      3. By default, the first argument that isn't an option of the java command is the fully qualified name of the class to be called.
      4. If -jar is specified,
         1. then its argument is the name of the JAR file containing class and resource files for the application.
         2. The startup class must be indicated by the Main-Class manifest header in its manifest file.
      5. Arguments after the class file name or the JAR file name are passed to the main() method.

### Synopsis

To launch a class file:

java [options] *mainclass* [args ...]

To launch the main class in a JAR file:

java [options] -jar *jarfile* [args ...]

To launch the main class in a module:

java [options] [-m|--module] *module*[/mainclass] [args ...]

To launch a single source-file program:

java [options] *source-file* [args ...]

|  |  |
| --- | --- |
| options |  |
| mainclass | Specifies the name of the class to be launched.  Command-line entries following classname are the arguments for the main method. |
| -jar jarfile | Executes a program encapsulated in a JAR file.  The jarfile argument is the name of a JAR file with a manifest that contains a line in the form Main-Class:classname that defines the class with the public static void main(String[] args) method that serves as your application's starting point.  When you use -jar, the specified JAR file is the source of all user classes, and other class path settings are ignored.  If you're using JAR files, then see jar. |
| -m | --module | Executes the main class in a module specified by mainclass if it is given, or, if it is not given, the value in the module.  In other words, mainclass can be used when it is not specified by the module, or to override the value when it is specified.  See Standard Options for Java. |
| source-file | Only used to launch a single source-file program. Specifies the source file that contains the main class when using source-file mode  See Using Source-File Mode to Launch Single-File Source-Code Programs |
| args... | Optional: Arguments following mainclass, source-file, -jar jarfile, and -m or --module module/mainclass are passed as arguments to the main class. |

### Using Source-File Mode to Launch Single-File Source-Code Programs

* + - 1. To launch a class declared in a source file, run the java launcher in source-file mode.
      2. Entering source-file mode is determined by two items on the java command line:
         1. The first item on the command line that is not an option or part of an option. In other words, the item in the command line that would otherwise be the main class name.
         2. The --source version option, if present.
      3. If the class identifies an existing file that has a .java extension, or if the --source option is specified, then source-file mode is selected.
      4. The source file is then compiled and run.
      5. The --source option can be used to specify the source version or N of the source code.
         1. This determines the API that can be used.
         2. When you set --source N, you can only use the public API that was defined in JDK N.
         3. The valid values of N change for each release, with new values added and old values removed.
         4. You'll get an error message if you use a value of N that is no longer supported.
         5. The supported values of N are the current Java SE release (15) and a limited number of previous releases, detailed in the command-line help for javac, under the --source and --release options.
      6. If the file does not have the .java extension, the --source option must be used to tell the java command to use the source-file mode.
      7. The --source option is used for cases when the source file is a "script" to be executed and the name of the source file does not follow the normal naming conventions for Java source files.
      8. In source-file mode, the effect is as though the source file is compiled into memory, and the first class found in the source file is executed.
      9. Any arguments placed after the name of the source file in the original command line are passed to the compiled class when it is executed.
         1. For example, if a file were named HelloWorld.java and contained a class named hello.World, then the source-file mode command to launch the class would be:

java HelloWorld.java

* + - * 1. The example illustrates that the class can be in a named package, and does not need to be in the unnamed package. This use of source-file mode is informally equivalent to using the following two commands where hello.World is the name of the class in the package:

javac -d <memory> HelloWorld.java

java -cp <memory> hello.World

* + - 1. See JEP 330: Launch Single-File Source-Code Programs for complete details.

### In source-file mode

* + - 1. any additional command-line options are processed as follows:
         1. The launcher scans the options specified before the source file for any that are relevant in order to compile the source file.This includes: --class-path, --module-path, --add-exports, --add-modules, --limit-modules, --patch-module, --upgrade-module-path, and any variant forms of those options. It also includes the new --enable-preview option, described in JEP 12.
         2. No provision is made to pass any additional options to the compiler, such as -processor or -Werror.
         3. Command-line argument files (@-files) may be used in the standard way. Long lists of arguments for either the VM or the program being invoked may be placed in files specified on the command-line by prefixing the filename with an @ character.
      2. compilation proceeds as follows:
         1. Any command-line options that are relevant to the compilation environment are taken into account.
         2. No other source files are found and compiled, as if the source path is set to an empty value.
         3. Annotation processing is disabled, as if -proc:none is in effect.
         4. If a version is specified, via the --source option, the value is used as the argument for an implicit --release option for the compilation. This sets both the source version accepted by compiler and the system API that may be used by the code in the source file.
         5. The source file is compiled in the context of an unnamed module.
         6. The source file should contain one or more top-level classes, the first of which is taken as the class to be executed.
         7. The compiler does not enforce the optional restriction defined at the end of JLS ??7.6, that a type in a named package should exist in a file whose name is composed from the type name followed by the .java extension.
         8. If the source file contains errors, appropriate error messages are written to the standard error stream, and the launcher exits with a non-zero exit code.
      3. execution proceeds as follows:
         1. The class to be executed is the first top-level class found in the source file. It must contain a declaration of the standard public static void main(String[]) method.
         2. The compiled classes are loaded by a custom class loader, that delegates to the application class loader. This implies that classes appearing on the application class path cannot refer to any classes declared in the source file.
         3. The compiled classes are executed in the context of an unnamed module, as though --add-modules=ALL-DEFAULT is in effect. This is in addition to any other --add-module options that may be have been specified on the command line.
         4. Any arguments appearing after the name of the file on the command line are passed to the standard main method in the obvious way.
         5. It is an error if there is a class on the application class path whose name is the same as that of the class to be executed.

### Overview of Java Option

* + - 1. The java command supports a wide range of options in the following categories:
         1. **Standard Options** for Java:

Options guaranteed to be supported by all implementations of the Java Virtual Machine (JVM).

They're used for common actions, such as checking the version of the JRE, setting the class path, enabling verbose output, and so on.

* + - * 1. **Extra Options** for Java: General purpose options that are specific to the Java **HotSpot** Virtual Machine. They aren't guaranteed to be supported by all JVM implementations, and are subject to change. These options start with -X.
      1. The advanced options aren't recommended for casual use.
         1. These are developer options used for tuning specific areas of the Java HotSpot Virtual Machine operation that often have specific system requirements and may require privileged access to system configuration parameters.
         2. Several examples of performance tuning are provided in Performance Tuning Examples.
         3. These options aren't guaranteed to be supported by all JVM implementations and are subject to change.
         4. Advanced options start with -XX.
         5. Category

Advanced Runtime Options for Java: Control the runtime behavior of the Java HotSpot VM.

Advanced JIT Compiler Options for java: Control the dynamic just-in-time (JIT) compilation performed by the Java HotSpot VM.

Advanced Serviceability Options for Java: Enable gathering system information and performing extensive debugging.

Advanced Garbage Collection Options for Java: Control how garbage collection (GC) is performed by the Java HotSpot

* + - 1. Boolean options are used to either enable a feature that's disabled by default or disable a feature that's enabled by default.
         1. Such options don't require a parameter.
         2. Boolean -XX options are enabled using the plus sign (-XX:+OptionName) and disabled using the minus sign (-XX:-OptionName).
      2. For options that require an argument, the argument may be separated from the option name by a space, a colon (:), or an equal sign (=), or the argument may directly follow the option (the exact syntax differs for each option).
      3. If you're expected to specify the size in bytes, then you can use no suffix, or use the suffix k or K for kilobytes (KB), m or M for megabytes (MB), or g or G for gigabytes (GB).

For example, to set the size to 8 GB, you can specify either 8g, 8192m, 8388608k, or 8589934592 as the argument.

* + - 1. If you are expected to specify the percentage, then use a number from 0 to 1. For example, specify 0.25 for 25%.
      2. The following sections describe the options that are obsolete, deprecated, and removed:
         1. Deprecated Java Options: Accepted and acted upon --- a warning is issued when they're used.
         2. Obsolete Java Options: Accepted but ignored --- a warning is issued when they're used.
         3. Removed Java Options: Removed --- using them results in an error.

### Standard Options for Java

These are the most commonly used options supported by all implementations of the JVM

|  |  |
| --- | --- |
| -agentlib | 1. Loads the specified **native agent library**. 2. After the library name, a comma-separated list of options specific to the library can be used. 3. Linux and macOS: If the option -agentlib:foo is specified, then the JVM attempts to load the library named libfoo.so in the location specified by the LD\_LIBRARY\_PATH system variable (on macOS this variable is DYLD\_LIBRARY\_PATH). 4. The following example shows how to load the Java Debug Wire Protocol (JDWP) library and listen for the socket connection on port 8000, suspending the JVM before the main class loads:   -agentlib:jdwp=transport=dt\_socket,server=y,address=8000 |
| -agentpath:pathname[=options] | 1. Loads the native agent library specified by the absolute path name. 2. This option is equivalent to -agentlib but uses the full path and file name of the library. |
| --class-path classpath,  -classpath classpath,  -cp classpath | 1. A semicolon (;) separated list of directories, JAR archives, and ZIP archives to search for class files. 2. Specifying classpath overrides any setting of the CLASSPATH environment variable. 3. If the class path option isn't used and classpath isn't set, then the user class path consists of the current directory (.). 4. As a special convenience, a class path element that contains a base name of an asterisk (\*) is considered equivalent to specifying a list of all the files in the directory with the extension .jar or .JAR . 5. A Java program can't tell the difference between the two invocations. 6. For example, if the directory mydir contains a.jar and b.JAR, then the class path element mydir/\* is expanded to A.jar:b.JAR, except that the order of JAR files is unspecified.    1. All .jar files in the specified directory, even hidden ones, are included in the list.    2. A class path entry consisting of an asterisk (\*) expands to a list of all the jar files in the current directory.    3. The CLASSPATH environment variable, where defined, is similarly expanded.    4. Any class path wildcard expansion that occurs before the Java VM is started.    5. Java programs never see wildcards that aren't expanded except by querying the environment, such as by calling System.getenv("CLASSPATH"). |
| --disable-@files | Can be used anywhere on the command line, including in an argument file, to prevent further @filename expansion.  This option stops expanding @-argfiles after the option. |
| --enable-preview | Allows classes to depend on preview features of the release. |
| --module-path modulepath...  -p modulepath | A semicolon (;) separated list of directories in which each directory is a directory of modules. |
| --upgrade-module-path modulepath... | A semicolon (;) separated list of directories in which each directory is a directory of modules that replace upgradeable modules in the runtime image. |
| --add-modules module[,module...] | Specifies the root modules to resolve in addition to the initial module.  module also can be ALL-DEFAULT, ALL-SYSTEM, and ALL-MODULE-PATH. |
| --list-modules | Lists the observable modules and then exits. |
| -d module\_name  --describe-module module\_name | Describes a specified module and then exits. |
| --dry-run | Creates the VM but doesn't execute the main method.  This --dry-run option might be useful for validating the command-line options such as the module system configuration. |
| --validate-modules | Validates all modules and exit.  This option is helpful for finding conflicts and other errors with modules on the module path. |
| -Dproperty=value | Sets a system property value.  The property variable is a string with no spaces that represents the name of the property.  The value variable is a string that represents the value of the property.  If value is a string with spaces, then enclose it in quotation marks (for example -Dfoo="foo bar"). |
| -disableassertions[:[packagename]...|:classname]  -da[:[packagename]...|:classname] | 1. Disables assertions. 2. By default, assertions are disabled in all packages and classes. 3. With no arguments, -disableassertions (-da) disables assertions in all packages and classes. 4. With the packagename argument ending in ..., the switch disables assertions in the specified package and any subpackages. 5. If the argument is simply ..., then the switch disables assertions in the unnamed package in the current working directory. 6. With the classname argument, the switch disables assertions in the specified class. 7. The -disableassertions (-da) option applies to all class loaders and to system classes (which don't have a class loader). There's one exception to this rule: If the option is provided with no arguments, then it doesn't apply to system classes. 8. This makes it easy to disable assertions in all classes except for system classes. 9. The -disablesystemassertions option enables you to disable assertions in all system classes. 10. To explicitly enable assertions in specific packages or classes, use the -enableassertions (-ea) option. 11. Both options can be used at the same time. 12. For example, to run the MyClass application with assertions enabled in the package com.wombat.fruitbat (and any subpackages) but disabled in the class com.wombat.fruitbat.Brickbat, use the following command:   java -ea:com.wombat.fruitbat... -da:com.wombat.fruitbat.Brickbat MyClass |
| -disablesystemassertions  -dsa | Disables assertions in all system classes. |
| -enableassertions[:[packagename]...|:classname]  -ea[:[packagename]...|:classname] | 1. Enables assertions. 2. By default, assertions are disabled in all packages and classes. 3. With no arguments, -enableassertions (-ea) enables assertions in all packages and classes. 4. With the packagename argument ending in ..., the switch enables assertions in the specified package and any subpackages. 5. If the argument is simply ..., then the switch enables assertions in the unnamed package in the current working directory. 6. With the classname argument, the switch enables assertions in the specified class. 7. The -enableassertions (-ea) option applies to all class loaders and to system classes (which don't have a class loader). 8. There's one exception to this rule: If the option is provided with no arguments, then it doesn't apply to system classes. 9. This makes it easy to enable assertions in all classes except for system classes. 10. The -enablesystemassertions option provides a separate switch to enable assertions in all system classes. 11. To explicitly disable assertions in specific packages or classes, use the -disableassertions (-da) option. 12. If a single command contains multiple instances of these switches, then they're processed in order, before loading any classes. 13. For example, to run the MyClass application with assertions enabled only in the package com.wombat.fruitbat (and any subpackages) but disabled in the class com.wombat.fruitbat.Brickbat, use the following command:   java -ea:com.wombat.fruitbat... -da:com.wombat.fruitbat.Brickbat MyClass |
| -enablesystemassertions  -esa | Enables assertions in all system classes. |
| -help  -h  -? | Prints the help message to the error stream. |
| --help | Prints the help message to the output stream. |
| -javaagent:jarpath[=options] | Loads the specified Java programming language agent. See java.lang.instrument. |
| --show-version | Prints the product version to the output stream and continues. |
| -showversion | Prints the product version to the error stream and continues. |
| --show-module-resolution | Shows module resolution output during startup. |
| -splash:imagepath | 1. Shows the splash screen with the image specified by imagepath. HiDPI scaled images are automatically supported and used if available. 2. The unscaled image file name, such as image.ext, should always be passed as the argument to the -splash option. The most appropriate scaled image provided is picked up automatically. 3. For example, to show the splash.gif file from the images directory when starting your application, use the following option:   -splash:images/splash.gif   1. See the SplashScreen API documentation for more information. |
| -verbose:class | Displays information about each loaded class. |
| -verbose:gc | Displays information about each garbage collection (GC) event. |
| -verbose:jni | Displays information about the use of native methods and other Java Native Interface (JNI) activity. |
| -verbose:module | Displays information about the modules in use. |
| --version | Prints product version to the output stream and exits. |
| -version | Prints product version to the error stream and exits. |
| --help-extra | Prints the help on extra options to the output stream. |
| @argfile | 1. Specifies one or more argument files prefixed by @ used by the java command. 2. It isn't uncommon for the java command line to be very long because of the .jar files needed in the classpath. 3. The @argfile option overcomes command-line length limitations by enabling the launcher to expand the contents of argument files after shell expansion, but before argument processing. 4. Contents in the argument files are expanded because otherwise, they would be specified on the command line until the --disable-@files option was encountered. 5. The argument files can also contain the main class name and all options. If an argument file contains all of the options required by the java command, then the command line could simply be:   java @argfile   1. See java Command-Line Argument Files for a description and examples of using @-argfiles. |

### Extra Options for Java

The following java options are general purpose options that are specific to the Java HotSpot Virtual Machine.

|  |  |
| --- | --- |
| -Xbatch | 1. Disables background compilation. 2. By default, the JVM compiles the method as a background task, running the method in interpreter mode until the background compilation is finished. 3. The -Xbatch flag disables background compilation so that compilation of all methods proceeds as a foreground task until completed. 4. This option is equivalent to -XX:-BackgroundCompilation. |
| -Xbootclasspath/a:directories|zip|JAR-files | 1. Specifies a list of directories, JAR files, and ZIP archives to append to the end of the default bootstrap class path. 2. Linux and macOS: Colons (:) separate entities in this list. |
| -Xcheck:jni | 1. Performs additional checks for Java Native Interface (JNI) functions. 2. Specifically, it validates the parameters passed to the JNI function and the runtime environment data before processing the JNI request. 3. It also checks for pending exceptions between JNI calls. 4. Any invalid data encountered indicates a problem in the native code, and the JVM terminates with an irrecoverable error in such cases. 5. Expect a performance degradation when this option is used. |
| -Xdebug | Does nothing. Provided for backward compatibility. |
| -Xdiag | Shows additional diagnostic messages. |
| -Xint | Runs the application in interpreted-only mode.  Compilation to native code is disabled, and all bytecode is executed by the interpreter.  The performance benefits offered by the just-in-time (JIT) compiler aren't present in this mode. |
| -Xinternalversion | Displays more detailed JVM version information than the -version option, and then exits. |
| -Xlog:option | 1. Configure or enable logging with the Java Virtual Machine (JVM) unified logging framework. 2. See Enable Logging with the JVM Unified Logging Framework. |
| -Xmixed | 1. Executes all bytecode by the interpreter except for hot methods, which are compiled to native code. 2. On by default. 3. Use -Xint to switch off. |
| -Xmn size | 1. Sets the initial and maximum size (in bytes) of the heap for the young generation (nursery) in the generational collectors. 2. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. 3. The young generation region of the heap is used for new objects. 4. GC is performed in this region more often than in other regions. 5. If the size for the young generation is too small, then a lot of minor garbage collections are performed. 6. If the size is too large, then only full garbage collections are performed, which can take a long time to complete. 7. It is recommended that you do not set the size for the young generation for the G1 collector, and keep the size for the young generation greater than 25% and less than 50% of the overall heap size for other collectors. 8. The following examples show how to set the initial and maximum size of young generation to 256 MB using various units:   -Xmn256m  -Xmn262144k  -Xmn268435456   1. Instead of the -Xmn option to set both the initial and maximum size of the heap for the young generation, you can use -XX:NewSize to set the initial size and -XX:MaxNewSize to set the maximum size. |
| -Xms size | 1. Sets the minimum and initial size (in bytes) of the heap. 2. This value must be a multiple of 1024 and greater than 1 MB. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, g or G to indicate gigabytes. 3. The following examples show how to set the size of allocated memory to 6 MB using various units:   -Xms6291456  -Xms6144k  -Xms6m   1. Instead of the -Xms option to set both the minimum and initial size of the heap, you can use -XX:MinHeapSize to set the minimum size and -XX:InitialHeapSize to set the initial size. 2. If you don't set this option, the initial size is set as the sum of the sizes allocated for the old generation and the young generation. 3. The initial size of the heap for the young generation can be set using the -Xmn option or the -XX:NewSize option. |
| -Xmx size | 1. Specifies the maximum size (in bytes) of the heap. This value must be a multiple of 1024 and greater than 2 MB. 2. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. 3. The default value is chosen at runtime based on system configuration. 4. For server deployments, -Xms and -Xmx are often set to the same value. 5. The following examples show how to set the maximum allowed size of allocated memory to 80 MB using various units:   -Xmx83886080  -Xmx81920k  -Xmx80m  The -Xmx option is equivalent to -XX:MaxHeapSize. |
| -Xnoclassgc | 1. Disables garbage collection (GC) of classes. 2. This can save some GC time, which shortens interruptions during the application run. 3. When you specify -Xnoclassgc at startup, the class objects in the application are left untouched during GC and are always be considered live. 4. This can result in more memory being permanently occupied which, if not used carefully, throws an out-of-memory exception. |
| -Xrs | 1. Reduces the use of operating system signals by the JVM. 2. Shutdown hooks enable the orderly shutdown of a Java application by running user cleanup code (such as closing database connections) at shutdown, even if the JVM terminates abruptly. 3. Linux and macOS:    1. The JVM catches signals to implement shutdown hooks for unexpected termination.    2. The JVM uses SIGHUP, SIGINT, and SIGTERM to initiate the running of shutdown hooks.    3. Applications embedding the JVM frequently need to trap signals such as SIGINT or SIGTERM, which can lead to interference with the JVM signal handlers.    4. The -Xrs option is available to address this issue.    5. When -Xrs is used, the signal masks for SIGINT, SIGTERM, SIGHUP, and SIGQUIT aren't changed by the JVM, and signal handlers for these signals aren't installed.    6. SIGQUIT thread dumps aren't available. 4. User code is responsible for causing shutdown hooks to run, for example, by calling the System.exit() when the JVM is to be terminated. |
| -Xshare:mode | 1. Sets the class data sharing (CDS) mode. 2. Possible mode arguments for this option include the following:   auto  Use shared class data if possible (default).  on  Require using shared class data, otherwise fail.  Note: The -Xshare:on option is used for testing purposes only and may cause intermittent failures due to the use of address space layout randomization by the operation system. This option should not be used in production environments.  off  Do not attempt to use shared class data. |
| -XshowSettings | Shows all settings and then continues. |
| -XshowSettings:category | Shows settings and continues.  Possible category arguments for this option include the following:  all  Shows all categories of settings. This is the default value.  locale  Shows settings related to locale.  properties  Shows settings related to system properties.  vm  Shows the settings of the JVM.  system  Linux: Shows host system or container configuration and continues. |
| -Xss size | 1. Sets the thread stack size (in bytes). 2. Append the letter k or K to indicate KB, m or M to indicate MB, or g or G to indicate GB. 3. The default value depends on the platform: Linux/x64 (64-bit): 1024 KB 4. This option is similar to -XX:ThreadStackSize. |
| --add-reads module=target-module(,target-module)\* | Updates module to read the target-module, regardless of the module declaration. target-module can be all unnamed to read all unnamed modules. |
| --add-exports module/package=target-module(,target-module)\* | Updates module to export package to target-module, regardless of module declaration. The target-module can be all unnamed to export to all unnamed modules. |
| --add-opens module/package=target-module(,target-module)\* | Updates module to open package to target-module, regardless of module declaration. |
| --illegal-access=parameter | 1. Note: This option will be removed in a future release. 2. When present at run time, --illegal-access= takes a keyword parameter to specify a mode of operation:   permit:  This mode opens each package in each module in the run-time image to code in all unnamed modules ( such as code on the class path), if that package existed in JDK 8.  This enables both static access, (for example, by compiled bytecode, and deep reflective access) through the platform's various reflection APIs.  The first reflective-access operation to any such package causes a warning to be issued.  However, no warnings are issued after the first occurrence.  This single warning describes how to enable further warnings.  This mode is the default for the current JDK but will change in a future release.  warn:  This mode is identical to permit except that a warning message is issued for each illegal reflective-access operation.  debug:  This mode is identical to warn except that both a warning message and a stack trace are issued for each illegal reflective-access operation.  deny:  This mode disables all illegal-access operations except for those enabled by other command-line options, such as --add-opens.  This mode will become the default in a future release.   1. The default mode, --illegal-access=permit, is intended to make you aware of code on the class path that reflectively accesses any JDK-internal APIs at least once. 2. To learn about all such accesses, you can use the warn or the debug modes. 3. For each library or framework on the class path that requires illegal access, you have two options:    1. If the component's maintainers have already released a fixed version that no longer uses JDK-internal APIs then you can consider upgrading to that version.    2. If the component still needs to be fixed, then you can contact its maintainers and ask them to replace their use of JDK-internal APIs with the proper exported APIs. 4. If you must continue to use a component that requires illegal access, then you can eliminate the warning messages by using one or more --add-opens options to open only those internal packages to which access is required. 5. To verify that your application is ready for a future version of the JDK, run it with --illegal-access=deny along with any necessary --add-opens options. 6. Any remaining illegal-access errors will most likely be due to static references from compiled code to JDK-internal APIs. 7. You can identify those by running the jdeps tool with the --jdk-internals option. 8. **For performance reasons**, the current JDK does not issue warnings for illegal static-access operations. |
| --limit-modules module[,module...] | Specifies the limit of the universe of observable modules. |
| --patch-module module=file(;file)\* | Overrides or augments a module with classes and resources in JAR files or directories. |
| --source version | Sets the version of the source in source-file mode. |

### Advanced Options for Java

These java options can be used to enable other advanced options.

|  |  |
| --- | --- |
| -XX:+UnlockDiagnosticVMOptions | 1. Unlocks the options intended for diagnosing the JVM. 2. By default, this option is disabled and diagnostic options aren't available. 3. Command line options that are enabled with the use of this option are not supported. 4. If you encounter issues while using any of these options, it is very likely that you will be required to reproduce the problem without using any of these unsupported options before Oracle Support can assist with an investigation. 5. It is also possible that any of these options may be removed or their behavior changed without any warning. |
| -XX:+UnlockExperimentalVMOptions | Unlocks the options that provide experimental features in the JVM. By default, this option is disabled and experimental features aren't available. |

### Advanced Runtime Options for Java

These java options control the runtime behavior of the Java HotSpot VM.

|  |  |
| --- | --- |
| -XX:ActiveProcessorCount=x | Overrides the number of CPUs that the VM will use to calculate the size of thread pools it will use for various operations such as Garbage Collection and ForkJoinPool.  The VM normally determines the number of available processors from the operating system. This flag can be useful for partitioning CPU resources when running multiple Java processes in docker containers. This flag is honored even if UseContainerSupport is not enabled. See -XX:-UseContainerSupport for a description of enabling and disabling container support. |
| -XX:AllocateHeapAt=path | Takes a path to the file system and uses memory mapping to allocate the object heap on the memory device. Using this option enables the HotSpot VM to allocate the Java object heap on an alternative memory device, such as an NV-DIMM, specified by the user.  Alternative memory devices that have the same semantics as DRAM, including the semantics of atomic operations, can be used instead of DRAM for the object heap without changing the existing application code. All other memory structures (such as the code heap, metaspace, and thread stacks) continue to reside in DRAM.  Some operating systems expose non-DRAM memory through the file system. Memory-mapped files in these file systems bypass the page cache and provide a direct mapping of virtual memory to the physical memory on the device. The existing heap related flags (such as -Xmx and -Xms) and garbage-collection related flags continue to work as before. |
| -XX:-CompactStrings | Disables the Compact Strings feature. By default, this option is enabled. When this option is enabled, Java Strings containing only single-byte characters are internally represented and stored as single-byte-per-character Strings using ISO-8859-1 / Latin-1 encoding. This reduces, by 50%, the amount of space required for Strings containing only single-byte characters. For Java Strings containing at least one multibyte character: these are represented and stored as 2 bytes per character using UTF-16 encoding. Disabling the Compact Strings feature forces the use of UTF-16 encoding as the internal representation for all Java Strings.  Cases where it may be beneficial to disable Compact Strings include the following:  When it's known that an application overwhelmingly will be allocating multibyte character Strings  In the unexpected event where a performance regression is observed in migrating from Java SE 8 to Java SE 9 and an analysis shows that Compact Strings introduces the regression  In both of these scenarios, disabling Compact Strings makes sense. |
| -XX:ErrorFile=filename | Specifies the path and file name to which error data is written when an irrecoverable error occurs. By default, this file is created in the current working directory and named hs\_err\_pidpid.log where pid is the identifier of the process that encountered the error.  The following example shows how to set the default log file (note that the identifier of the process is specified as %p):  -XX:ErrorFile=./hs\_err\_pid%p.log  Linux and macOS: The following example shows how to set the error log to /var/log/java/java\_error.log:  -XX:ErrorFile=/var/log/java/java\_error.log  Windows: The following example shows how to set the error log file to C:/log/java/java\_error.log:  -XX:ErrorFile=C:/log/java/java\_error.log  If the file exists, and is writeable, then it will be overwritten. Otherwise, if the file can't be created in the specified directory (due to insufficient space, permission problem, or another issue), then the file is created in the temporary directory for the operating system:  Linux and macOS: The temporary directory is /tmp.  Windows: The temporary directory is specified by the value of the TMP environment variable; if that environment variable isn't defined, then the value of the TEMP environment variable is used. |
| -XX:+ExtensiveErrorReports | Enables the reporting of more extensive error information in the ErrorFile. This option can be turned on in environments where maximal information is desired - even if the resulting logs may be quite large and/or contain information that might be considered sensitive. The information can vary from release to release, and across different platforms. By default this option is disabled. |
| -XX:FlightRecorderOptions=parameter=value  -XX:FlightRecorderOptions:parameter=value | Sets the parameters that control the behavior of JFR.  The following list contains the available JFR parameter=value entries:  globalbuffersize=size  Specifies the total amount of primary memory used for data retention. The default value is based on the value specified for memorysize. Change the memorysize parameter to alter the size of global buffers.  maxchunksize=size  Specifies the maximum size (in bytes) of the data chunks in a recording. Append m or M to specify the size in megabytes (MB), or g or G to specify the size in gigabytes (GB). By default, the maximum size of data chunks is set to 12 MB. The minimum allowed is 1 MB.  memorysize=size  Determines how much buffer memory should be used, and sets the globalbuffersize and numglobalbuffers parameters based on the size specified. Append m or M to specify the size in megabytes (MB), or g or G to specify the size in gigabytes (GB). By default, the memory size is set to 10 MB.  numglobalbuffers  Specifies the number of global buffers used. The default value is based on the memory size specified. Change the memorysize parameter to alter the number of global buffers.  old-object-queue-size=number-of-objects  Maximum number of old objects to track. By default, the number of objects is set to 256.  repository=path  Specifies the repository (a directory) for temporary disk storage. By default, the system's temporary directory is used.  retransform={true|false}  Specifies whether event classes should be retransformed using JVMTI. If false, instrumentation is added when event classes are loaded. By default, this parameter is enabled.  samplethreads={true|false}  Specifies whether thread sampling is enabled. Thread sampling occurs only if the sampling event is enabled along with this parameter. By default, this parameter is enabled.  stackdepth=depth  Stack depth for stack traces. By default, the depth is set to 64 method calls. The maximum is 2048. Values greater than 64 could create significant overhead and reduce performance.  threadbuffersize=size  Specifies the per-thread local buffer size (in bytes). By default, the local buffer size is set to 8 kilobytes, with a minimum value of 4 kilobytes. Overriding this parameter could reduce performance and is not recommended.  You can specify values for multiple parameters by separating them with a comma. |
| -XX:LargePageSizeInBytes=size | Sets the maximum size (in bytes) for large pages used for the Java heap. The size argument must be a power of 2 (2, 4, 8, 16, and so on). Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. By default, the size is set to 0, meaning that the JVM chooses the size for large pages automatically. See Large Pages.  The following example describes how to set the large page size to 4 megabytes (MB):  -XX:LargePageSizeInBytes=4m |
| -XX:MaxDirectMemorySize=size | Sets the maximum total size (in bytes) of the java.nio package, direct-buffer allocations. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. By default, the size is set to 0, meaning that the JVM chooses the size for NIO direct-buffer allocations automatically.  The following examples illustrate how to set the NIO size to 1024 KB in different units:  -XX:MaxDirectMemorySize=1m  -XX:MaxDirectMemorySize=1024k  -XX:MaxDirectMemorySize=1048576 |
| -XX:-MaxFDLimit | Disables the attempt to set the soft limit for the number of open file descriptors to the hard limit. By default, this option is enabled on all platforms, but is ignored on Windows. The only time that you may need to disable this is on Mac OS, where its use imposes a maximum of 10240, which is lower than the actual system maximum. |
| -XX:NativeMemoryTracking=mode | Specifies the mode for tracking JVM native memory usage. Possible mode arguments for this option include the following:  off  Instructs not to track JVM native memory usage. This is the default behavior if you don't specify the -XX:NativeMemoryTracking option.  summary  Tracks memory usage only by JVM subsystems, such as Java heap, class, code, and thread.  detail  In addition to tracking memory usage by JVM subsystems, track memory usage by individual CallSite, individual virtual memory region and its committed regions. |
| -XX:ObjectAlignmentInBytes=alignment | Sets the memory alignment of Java objects (in bytes). By default, the value is set to 8 bytes. The specified value should be a power of 2, and must be within the range of 8 and 256 (inclusive). This option makes it possible to use compressed pointers with large Java heap sizes.  The heap size limit in bytes is calculated as:  4GB \* ObjectAlignmentInBytes  Note: As the alignment value increases, the unused space between objects also increases. As a result, you may not realize any benefits from using compressed pointers with large Java heap sizes. |
| -XX:OnError=string | Sets a custom command or a series of semicolon-separated commands to run when an irrecoverable error occurs. If the string contains spaces, then it must be enclosed in quotation marks.  Linux and macOS: The following example shows how the -XX:OnError option can be used to run the gcore command to create a core image, and start the gdb debugger to attach to the process in case of an irrecoverable error (the %p designates the current process identifier):  -XX:OnError="gcore %p;gdb -p %p"  Windows: The following example shows how the -XX:OnError option can be used to run the userdump.exe utility to obtain a crash dump in case of an irrecoverable error (the %p designates the current process identifier). This example assumes that the path to the userdump.exe utility is specified in the PATH environment variable:  -XX:OnError="userdump.exe %p" |
| -XX:OnOutOfMemoryError=string | Sets a custom command or a series of semicolon-separated commands to run when an OutOfMemoryError exception is first thrown. If the string contains spaces, then it must be enclosed in quotation marks. For an example of a command string, see the description of the -XX:OnError option. |
| -XX:+PrintCommandLineFlags | Enables printing of ergonomically selected JVM flags that appeared on the command line. It can be useful to know the ergonomic values set by the JVM, such as the heap space size and the selected garbage collector. By default, this option is disabled and flags aren't printed. |
| -XX:+PreserveFramePointer | Selects between using the RBP register as a general purpose register (-XX:-PreserveFramePointer) and using the RBP register to hold the frame pointer of the currently executing method (-XX:+PreserveFramePointer . If the frame pointer is available, then external profiling tools (for example, Linux perf) can construct more accurate stack traces. |
| -XX:+PrintNMTStatistics | Enables printing of collected native memory tracking data at JVM exit when native memory tracking is enabled (see -XX:NativeMemoryTracking). By default, this option is disabled and native memory tracking data isn't printed. |
| -XX:SharedArchiveFile=path | Specifies the path and name of the class data sharing (CDS) archive file  See Application Class Data Sharing. |
| -XX:SharedArchiveConfigFile=shared\_config\_file | Specifies additional shared data added to the archive file. |
| -XX:SharedClassListFile=file\_name | Specifies the text file that contains the names of the classes to store in the class data sharing (CDS) archive. This file contains the full name of one class per line, except slashes (/) replace dots (.). For example, to specify the classes java.lang.Object and hello.Main, create a text file that contains the following two lines:  java/lang/Object  hello/Main  The classes that you specify in this text file should include the classes that are commonly used by the application. They may include any classes from the application, extension, or bootstrap class paths.  See Application Class Data Sharing. |
| -XX:+ShowCodeDetailsInExceptionMessages | Enables printing of improved NullPointerException messages. When an application throws a NullPointerException, the option enables the JVM to analyze the program's bytecode instructions to determine precisely which reference is null, and describes the source with a null-detail message. The null-detail message is calculated and returned by NullPointerException.getMessage(), and will be printed as the exception message along with the method, filename, and line number. By default, this option is enabled. |
| -XX:+ShowMessageBoxOnError | Enables the display of a dialog box when the JVM experiences an irrecoverable error. This prevents the JVM from exiting and keeps the process active so that you can attach a debugger to it to investigate the cause of the error. By default, this option is disabled. |
| -XX:StartFlightRecording=parameter=value | Starts a JFR recording for the Java application. This option is equivalent to the JFR.start diagnostic command that starts a recording during runtime. You can set the following parameter=value entries when starting a JFR recording:  delay=time  Specifies the delay between the Java application launch time and the start of the recording. Append s to specify the time in seconds, m for minutes, h for hours, or d for days (for example, specifying 10m means 10 minutes). By default, there's no delay, and this parameter is set to 0.  disk={true|false}  Specifies whether to write data to disk while recording. By default, this parameter is enabled.  dumponexit={true|false}  Specifies if the running recording is dumped when the JVM shuts down. If enabled and a filename is not entered, the recording is written to a file in the directory where the process was started. The file name is a system-generated name that contains the process ID, recording ID, and current timestamp, similar to hotspot-pid-47496-id-1-2018\_01\_25\_19\_10\_41.jfr. By default, this parameter is disabled.  duration=time  Specifies the duration of the recording. Append s to specify the time in seconds, m for minutes, h for hours, or d for days (for example, specifying 5h means 5 hours). By default, the duration isn't limited, and this parameter is set to 0.  filename=path  Specifies the path and name of the file to which the recording is written when the recording is stopped, for example:  recording.jfr  /home/user/recordings/recording.jfr  c:\recordings\recording.jfr  name=identifier  Takes both the name and the identifier of a recording.  maxage=time  Specifies the maximum age of disk data to keep for the recording. This parameter is valid only when the disk parameter is set to true. Append s to specify the time in seconds, m for minutes, h for hours, or d for days (for example, specifying 30s means 30 seconds). By default, the maximum age isn't limited, and this parameter is set to 0s.  maxsize=size  Specifies the maximum size (in bytes) of disk data to keep for the recording. This parameter is valid only when the disk parameter is set to true. The value must not be less than the value for the maxchunksize parameter set with -XX:FlightRecorderOptions. Append m or M to specify the size in megabytes, or g or G to specify the size in gigabytes. By default, the maximum size of disk data isn't limited, and this parameter is set to 0.  path-to-gc-roots={true|false}  Specifies whether to collect the path to garbage collection (GC) roots at the end of a recording. By default, this parameter is disabled.  The path to GC roots is useful for finding memory leaks, but collecting it is time-consuming. Enable this option only when you start a recording for an application that you suspect has a memory leak. If the settings parameter is set to profile, the stack trace from where the potential leaking object was allocated is included in the information collected.  settings=path  Specifies the path and name of the event settings file (of type JFC). By default, the default.jfc file is used, which is located in JAVA\_HOME/lib/jfr. This default settings file collects a predefined set of information with low overhead, so it has minimal impact on performance and can be used with recordings that run continuously.  A second settings file is also provided, profile.jfc, which provides more data than the default configuration, but can have more overhead and impact performance. Use this configuration for short periods of time when more information is needed.  You can specify values for multiple parameters by separating them with a comma. |
| -XX:ThreadStackSize=size | Sets the Java thread stack size (in kilobytes). Use of a scaling suffix, such as k, results in the scaling of the kilobytes value so that -XX:ThreadStackSize=1k sets the Java thread stack size to 1024\*1024 bytes or 1 megabyte. The default value depends on the platform:  Linux/x64 (64-bit): 1024 KB  macOS (64-bit): 1024 KB  Windows: The default value depends on virtual memory  The following examples show how to set the thread stack size to 1 megabyte in different units:  -XX:ThreadStackSize=1k  -XX:ThreadStackSize=1024  This option is similar to -Xss. |
| -XX:-UseCompressedOops | Disables the use of compressed pointers. By default, this option is enabled, and compressed pointers are used. This will automatically limit the maximum ergonomically determined Java heap size to the maximum amount of memory that can be covered by compressed pointers. By default this range is 32 GB.  With compressed oops enabled, object references are represented as 32-bit offsets instead of 64-bit pointers, which typically increases performance when running the application with Java heap sizes smaller than the compressed oops pointer range. This option works only for 64-bit JVMs.  It's possible to use compressed pointers with Java heap sizes greater than 32 GB. See the -XX:ObjectAlignmentInBytes option. |
| -XX:-UseContainerSupport | The VM now provides automatic container detection support, which allows the VM to determine the amount of memory and number of processors that are available to a Java process running in docker containers. It uses this information to allocate system resources. This support is only available on Linux x64 platforms. If supported, the default for this flag is true, and container support is enabled by default. It can be disabled with -XX:-UseContainerSupport.  Unified Logging is available to help to diagnose issues related to this support.  Use -Xlog:os+container=trace for maximum logging of container information. See Enable Logging with the JVM Unified Logging Framework for a description of using Unified Logging. |
| -XX:+UseHugeTLBFS | Linux only: This option is the equivalent of specifying -XX:+UseLargePages. This option is disabled by default. This option pre-allocates all large pages up-front, when memory is reserved; consequently the JVM can't dynamically grow or shrink large pages memory areas; see -XX:UseTransparentHugePages if you want this behavior.  See Large Pages. |
| -XX:+UseLargePages | Enables the use of large page memory. By default, this option is disabled and large page memory isn't used.  See Large Pages. |
| -XX:+UseTransparentHugePages | Linux only: Enables the use of large pages that can dynamically grow or shrink. This option is disabled by default. You may encounter performance problems with transparent huge pages as the OS moves other pages around to create huge pages; this option is made available for experimentation. |
| -XX:+AllowUserSignalHandlers | Enables installation of signal handlers by the application. By default, this option is disabled and the application isn't allowed to install signal handlers. |
| -XX:VMOptionsFile=filename | Allows user to specify VM options in a file, for example, java -XX:VMOptionsFile=/var/my\_vm\_options HelloWorld. |

### Advanced JIT Compiler Options for java

These java options control the dynamic just-in-time (JIT) compilation performed by the Java HotSpot VM.

|  |  |
| --- | --- |
| -XX:AllocateInstancePrefetchLines=lines | Sets the number of lines to prefetch ahead of the instance allocation pointer. By default, the number of lines to prefetch is set to 1:  -XX:AllocateInstancePrefetchLines=1 |
| -XX:AllocatePrefetchDistance=size | Sets the size (in bytes) of the prefetch distance for object allocation. Memory about to be written with the value of new objects is prefetched up to this distance starting from the address of the last allocated object. Each Java thread has its own allocation point.  Negative values denote that prefetch distance is chosen based on the platform. Positive values are bytes to prefetch. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value is set to -1.  The following example shows how to set the prefetch distance to 1024 bytes:  -XX:AllocatePrefetchDistance=1024 |
| -XX:AllocatePrefetchInstr=instruction | Sets the prefetch instruction to prefetch ahead of the allocation pointer. Possible values are from 0 to 3. The actual instructions behind the values depend on the platform. By default, the prefetch instruction is set to 0:  -XX:AllocatePrefetchInstr=0 |
| -XX:AllocatePrefetchLines=lines | Sets the number of cache lines to load after the last object allocation by using the prefetch instructions generated in compiled code. The default value is 1 if the last allocated object was an instance, and 3 if it was an array.  The following example shows how to set the number of loaded cache lines to 5:  -XX:AllocatePrefetchLines=5 |
| -XX:AllocatePrefetchStepSize=size | Sets the step size (in bytes) for sequential prefetch instructions. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, g or G to indicate gigabytes. By default, the step size is set to 16 bytes:  -XX:AllocatePrefetchStepSize=16 |
| -XX:AllocatePrefetchStyle=style | Sets the generated code style for prefetch instructions. The style argument is an integer from 0 to 3:  0  Don't generate prefetch instructions.  1  Execute prefetch instructions after each allocation. This is the default setting.  2  Use the thread-local allocation block (TLAB) watermark pointer to determine when prefetch instructions are executed.  3  Generate one prefetch instruction per cache line. |
| -XX:+BackgroundCompilation | Enables background compilation. This option is enabled by default. To disable background compilation, specify -XX:-BackgroundCompilation (this is equivalent to specifying -Xbatch). |
| -XX:CICompilerCount=threads | Sets the number of compiler threads to use for compilation. By default, the number of compiler threads is selected automatically depending on the number of CPUs and memory available for compiled code. The following example shows how to set the number of threads to 2:  -XX:CICompilerCount=2 |
| -XX:+UseDynamicNumberOfCompilerThreads | Dynamically create compiler thread up to the limit specified by -XX:CICompilerCount. This option is enabled by default. |
| -XX:CompileCommand=command,method[,option] | Specifies a command to perform on a method. For example, to exclude the indexOf() method of the String class from being compiled, use the following:  -XX:CompileCommand=exclude,java/lang/String.indexOf  Note that the full class name is specified, including all packages and subpackages separated by a slash (/). For easier cut-and-paste operations, it's also possible to use the method name format produced by the -XX:+PrintCompilation and -XX:+LogCompilation options:  -XX:CompileCommand=exclude,java.lang.String::indexOf  If the method is specified without the signature, then the command is applied to all methods with the specified name. However, you can also specify the signature of the method in the class file format. In this case, you should enclose the arguments in quotation marks, because otherwise the shell treats the semicolon as a command end. For example, if you want to exclude only the indexOf(String) method of the String class from being compiled, use the following:  -XX:CompileCommand="exclude,java/lang/String.indexOf,(Ljava/lang/String;)I"  You can also use the asterisk (\*) as a wildcard for class and method names. For example, to exclude all indexOf() methods in all classes from being compiled, use the following:  -XX:CompileCommand=exclude,\*.indexOf  The commas and periods are aliases for spaces, making it easier to pass compiler commands through a shell. You can pass arguments to -XX:CompileCommand using spaces as separators by enclosing the argument in quotation marks:  -XX:CompileCommand="exclude java/lang/String indexOf"  Note that after parsing the commands passed on the command line using the -XX:CompileCommand options, the JIT compiler then reads commands from the .hotspot\_compiler file. You can add commands to this file or specify a different file using the -XX:CompileCommandFile option.  To add several commands, either specify the -XX:CompileCommand option multiple times, or separate each argument with the new line separator (\n). The following commands are available:  break  Sets a breakpoint when debugging the JVM to stop at the beginning of compilation of the specified method.  compileonly  Excludes all methods from compilation except for the specified method. As an alternative, you can use the -XX:CompileOnly option, which lets you specify several methods.  dontinline  Prevents inlining of the specified method.  exclude  Excludes the specified method from compilation.  help  Prints a help message for the -XX:CompileCommand option.  inline  Attempts to inline the specified method.  log  Excludes compilation logging (with the -XX:+LogCompilation option) for all methods except for the specified method. By default, logging is performed for all compiled methods.  option  Passes a JIT compilation option to the specified method in place of the last argument (option). The compilation option is set at the end, after the method name. For example, to enable the BlockLayoutByFrequency option for the append() method of the StringBuffer class, use the following:  -XX:CompileCommand=option,java/lang/StringBuffer.append,BlockLayoutByFrequency  You can specify multiple compilation options, separated by commas or spaces.  print  Prints generated assembler code after compilation of the specified method.  quiet  Instructs not to print the compile commands. By default, the commands that you specify with the -XX:CompileCommand option are printed; for example, if you exclude from compilation the indexOf() method of the String class, then the following is printed to standard output:  CompilerOracle: exclude java/lang/String.indexOf  You can suppress this by specifying the -XX:CompileCommand=quiet option before other -XX:CompileCommand options. |
| -XX:CompileCommandFile=filename | Sets the file from which JIT compiler commands are read. By default, the .hotspot\_compiler file is used to store commands performed by the JIT compiler.  Each line in the command file represents a command, a class name, and a method name for which the command is used. For example, this line prints assembly code for the toString() method of the String class:  print java/lang/String toString  If you're using commands for the JIT compiler to perform on methods, then see the -XX:CompileCommand option. |
| -XX:CompilerDirectivesFile=file | Adds directives from a file to the directives stack when a program starts. See Compiler Control.  The -XX:CompilerDirectivesFile option has to be used together with the -XX:UnlockDiagnosticVMOptions option that unlocks diagnostic JVM options. |
| -XX:+CompilerDirectivesPrint | Prints the directives stack when the program starts or when a new directive is added.  The -XX:+CompilerDirectivesPrint option has to be used together with the -XX:UnlockDiagnosticVMOptions option that unlocks diagnostic JVM options. |
| -XX:CompileOnly=methods | Sets the list of methods (separated by commas) to which compilation should be restricted. Only the specified methods are compiled. Specify each method with the full class name (including the packages and subpackages). For example, to compile only the length() method of the String class and the size() method of the List class, use the following:  -XX:CompileOnly=java/lang/String.length,java/util/List.size  Note that the full class name is specified, including all packages and subpackages separated by a slash (/). For easier cut and paste operations, it's also possible to use the method name format produced by the -XX:+PrintCompilation and -XX:+LogCompilation options:  -XX:CompileOnly=java.lang.String::length,java.util.List::size  Although wildcards aren't supported, you can specify only the class or package name to compile all methods in that class or package, as well as specify just the method to compile methods with this name in any class:  -XX:CompileOnly=java/lang/String  -XX:CompileOnly=java/lang  -XX:CompileOnly=.length |
| -XX:CompileThresholdScaling=scale | Provides unified control of first compilation. This option controls when methods are first compiled for both the tiered and the nontiered modes of operation. The CompileThresholdScaling option has a floating point value between 0 and +Inf and scales the thresholds corresponding to the current mode of operation (both tiered and nontiered). Setting CompileThresholdScaling to a value less than 1.0 results in earlier compilation while values greater than 1.0 delay compilation. Setting CompileThresholdScaling to 0 is equivalent to disabling compilation. |
| -XX:+DoEscapeAnalysis | Enables the use of escape analysis. This option is enabled by default. To disable the use of escape analysis, specify -XX:-DoEscapeAnalysis. |
| -XX:InitialCodeCacheSize=size | Sets the initial code cache size (in bytes). Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value depends on the platform. The initial code cache size shouldn't be less than the system's minimal memory page size. The following example shows how to set the initial code cache size to 32 KB:  -XX:InitialCodeCacheSize=32k |
| -XX:+Inline | Enables method inlining. This option is enabled by default to increase performance. To disable method inlining, specify -XX:-Inline. |
| -XX:InlineSmallCode=size | Sets the maximum code size (in bytes) for already compiled methods that may be inlined. This flag only applies to the C2 compiler. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value depends on the platform and on whether tiered compilation is enabled. In the following example it is set to 1000 bytes:  -XX:InlineSmallCode=1000 |
| -XX:+LogCompilation | Enables logging of compilation activity to a file named hotspot.log in the current working directory. You can specify a different log file path and name using the -XX:LogFile option.  By default, this option is disabled and compilation activity isn't logged. The -XX:+LogCompilation option has to be used together with the -XX:UnlockDiagnosticVMOptions option that unlocks diagnostic JVM options.  You can enable verbose diagnostic output with a message printed to the console every time a method is compiled by using the -XX:+PrintCompilation option. |
| -XX:FreqInlineSize=size | Sets the maximum bytecode size (in bytes) of a hot method to be inlined. This flag only applies to the C2 compiler. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value depends on the platform. In the following example it is set to 325 bytes:  -XX:FreqInlineSize=325 |
| -XX:MaxInlineSize=size | Sets the maximum bytecode size (in bytes) of a cold method to be inlined. This flag only applies to the C2 compiler. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. By default, the maximum bytecode size is set to 35 bytes:  -XX:MaxInlineSize=35 |
| -XX:C1MaxInlineSize=size | Sets the maximum bytecode size (in bytes) of a cold method to be inlined. This flag only applies to the C1 compiler. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. By default, the maximum bytecode size is set to 35 bytes:  -XX:MaxInlineSize=35 |
| -XX:MaxTrivialSize=size | Sets the maximum bytecode size (in bytes) of a trivial method to be inlined. This flag only applies to the C2 compiler. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. By default, the maximum bytecode size of a trivial method is set to 6 bytes:  -XX:MaxTrivialSize=6 |
| -XX:C1MaxTrivialSize=size | Sets the maximum bytecode size (in bytes) of a trivial method to be inlined. This flag only applies to the C1 compiler. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. By default, the maximum bytecode size of a trivial method is set to 6 bytes:  -XX:MaxTrivialSize=6 |
| -XX:MaxNodeLimit=nodes | Sets the maximum number of nodes to be used during single method compilation. By default the value depends on the features enabled. In the following example the maximum number of nodes is set to 100,000:  -XX:MaxNodeLimit=100000 |
| -XX:NonNMethodCodeHeapSize=size | Sets the size in bytes of the code segment containing nonmethod code.  A nonmethod code segment containing nonmethod code, such as compiler buffers and the bytecode interpreter. This code type stays in the code cache forever. This flag is used only if -XX:SegmentedCodeCache is enabled. |
| -XX:NonProfiledCodeHeapSize=size | Sets the size in bytes of the code segment containing nonprofiled methods. This flag is used only if -XX:SegmentedCodeCache is enabled. |
| -XX:+OptimizeStringConcat | Enables the optimization of String concatenation operations. This option is enabled by default. To disable the optimization of String concatenation operations, specify -XX:-OptimizeStringConcat. |
| -XX:+PrintAssembly | Enables printing of assembly code for bytecoded and native methods by using the external hsdis-<arch>.so or .dll library. For 64-bit VM on Windows, it's hsdis-amd64.dll. This lets you to see the generated code, which may help you to diagnose performance issues.  By default, this option is disabled and assembly code isn't printed. The -XX:+PrintAssembly option has to be used together with the -XX:UnlockDiagnosticVMOptions option that unlocks diagnostic JVM options. |
| -XX:ProfiledCodeHeapSize=size | Sets the size in bytes of the code segment containing profiled methods. This flag is used only if -XX:SegmentedCodeCache is enabled. |
| -XX:+PrintCompilation | Enables verbose diagnostic output from the JVM by printing a message to the console every time a method is compiled. This lets you to see which methods actually get compiled. By default, this option is disabled and diagnostic output isn't printed.  You can also log compilation activity to a file by using the -XX:+LogCompilation option. |
| -XX:+PrintInlining | Enables printing of inlining decisions. This let's you see which methods are getting inlined.  By default, this option is disabled and inlining information isn't printed. The -XX:+PrintInlining option has to be used together with the -XX:+UnlockDiagnosticVMOptions option that unlocks diagnostic JVM options. |
| -XX:ReservedCodeCacheSize=size | Sets the maximum code cache size (in bytes) for JIT-compiled code. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default maximum code cache size is 240 MB; if you disable tiered compilation with the option -XX:-TieredCompilation, then the default size is 48 MB. This option has a limit of 2 GB; otherwise, an error is generated. The maximum code cache size shouldn't be less than the initial code cache size; see the option -XX:InitialCodeCacheSize. |
| -XX:RTMAbortRatio=abort\_ratio | Specifies the RTM abort ratio is specified as a percentage (%) of all executed RTM transactions. If a number of aborted transactions becomes greater than this ratio, then the compiled code is deoptimized. This ratio is used when the -XX:+UseRTMDeopt option is enabled. The default value of this option is 50. This means that the compiled code is deoptimized if 50% of all transactions are aborted. |
| -XX:RTMRetryCount=number\_of\_retries | Specifies the number of times that the RTM locking code is retried, when it is aborted or busy, before falling back to the normal locking mechanism. The default value for this option is 5. The -XX:UseRTMLocking option must be enabled. |
| -XX:+SegmentedCodeCache | Enables segmentation of the code cache. Without the -XX:+SegmentedCodeCache, the code cache consists of one large segment. With -XX:+SegmentedCodeCache, we have separate segments for nonmethod, profiled method, and nonprofiled method code. These segments aren't resized at runtime. The feature is enabled by default if tiered compilation is enabled (-XX:+TieredCompilation ) and -XX:ReservedCodeCacheSize >= 240 MB. The advantages are better control of the memory footprint, reduced code fragmentation, and better iTLB/iCache behavior due to improved locality. iTLB/iCache is a CPU-specific term meaning Instruction Translation Lookaside Buffer (ITLB). ICache is an instruction cache in theCPU. The implementation of the code cache can be found in the file: /share/vm/code/codeCache.cpp. |
| -XX:StartAggressiveSweepingAt=percent | Forces stack scanning of active methods to aggressively remove unused code when only the given percentage of the code cache is free. The default value is 10%. |
| -XX:-TieredCompilation | Disables the use of tiered compilation. By default, this option is enabled. |
| -XX:UseSSE=version | Enables the use of SSE instruction set of a specified version. Is set by default to the highest supported version available (x86 only). |
| -XX:UseAVX=version | Enables the use of AVX instruction set of a specified version. Is set by default to the highest supported version available (x86 only). |
| -XX:+UseAES | Enables hardware-based AES intrinsics for hardware that supports it. This option is on by default on hardware that has the necessary instructions. The -XX:+UseAES is used in conjunction with UseAESIntrinsics. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseAESIntrinsics | Enables AES intrinsics. Specifying-XX:+UseAESIntrinsics is equivalent to also enabling -XX:+UseAES. To disable hardware-based AES intrinsics, specify -XX:-UseAES -XX:-UseAESIntrinsics. For example, to enable hardware AES, use the following flags:  -XX:+UseAES -XX:+UseAESIntrinsics  Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseAESCTRIntrinsics | Analogous to -XX:+UseAESIntrinsics enables AES/CTR intrinsics. |
| -XX:+UseGHASHIntrinsics | Controls the use of GHASH intrinsics. Enabled by default on platforms that support the corresponding instructions. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseBASE64Intrinsics | Controls the use of accelerated BASE64 encoding routines for java.util.Base64. Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseAdler32Intrinsics | Controls the use of Adler32 checksum algorithm intrinsic for java.util.zip.Adler32. Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseCRC32Intrinsics | Controls the use of CRC32 intrinsics for java.util.zip.CRC32. Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseCRC32CIntrinsics | Controls the use of CRC32C intrinsics for java.util.zip.CRC32C. Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseSHA | Enables hardware-based intrinsics for SHA crypto hash functions for some hardware. The UseSHA option is used in conjunction with the UseSHA1Intrinsics, UseSHA256Intrinsics, and UseSHA512Intrinsics options.  The UseSHA and UseSHA\*Intrinsics flags are enabled by default on machines that support the corresponding instructions.  This feature is applicable only when using the sun.security.provider.Sun provider for SHA operations. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions.  To disable all hardware-based SHA intrinsics, specify the -XX:-UseSHA. To disable only a particular SHA intrinsic, use the appropriate corresponding option. For example: -XX:-UseSHA256Intrinsics. |
| -XX:+UseSHA1Intrinsics | Enables intrinsics for SHA-1 crypto hash function. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseSHA256Intrinsics | Enables intrinsics for SHA-224 and SHA-256 crypto hash functions. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseSHA512Intrinsics | Enables intrinsics for SHA-384 and SHA-512 crypto hash functions. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseMathExactIntrinsics | Enables intrinsification of various java.lang.Math.\*Exact() functions. Enabled by default. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseMultiplyToLenIntrinsic | Enables intrinsification of BigInteger.multiplyToLen(). Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseSquareToLenIntrinsic | Enables intrinsification of BigInteger.squareToLen(). Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseMulAddIntrinsic | Enables intrinsification of BigInteger.mulAdd(). Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseMontgomeryMultiplyIntrinsic | Enables intrinsification of BigInteger.montgomeryMultiply(). Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseMontgomerySquareIntrinsic | Enables intrinsification of BigInteger.montgomerySquare(). Enabled by default on platforms that support it. Flags that control intrinsics now require the option -XX:+UnlockDiagnosticVMOptions. |
| -XX:+UseCMoveUnconditionally | Generates CMove (scalar and vector) instructions regardless of profitability analysis. |
| -XX:+UseCodeCacheFlushing | Enables flushing of the code cache before shutting down the compiler. This option is enabled by default. To disable flushing of the code cache before shutting down the compiler, specify -XX:-UseCodeCacheFlushing. |
| -XX:+UseCondCardMark | Enables checking if the card is already marked before updating the card table. This option is disabled by default. It should be used only on machines with multiple sockets, where it increases the performance of Java applications that rely on concurrent operations. |
| -XX:+UseCountedLoopSafepoints | Keeps safepoints in counted loops. Its default value depends on whether the selected garbage collector requires low latency safepoints. |
| -XX:LoopStripMiningIter=number | Controls the number of iterations in the inner strip mined loop. Strip mining transforms counted loops into two level nested loops. Safepoints are kept in the outer loop while the inner loop can execute at full speed. This option controls the maximum number of iterations in the inner loop. The default value is 1,000. |
| -XX:LoopStripMiningIterShortLoop=number | Controls loop strip mining optimization. Loops with the number of iterations less than specified will not have safepoints in them. Default value is 1/10th of -XX:LoopStripMiningIter. |
| -XX:+UseFMA | Enables hardware-based FMA intrinsics for hardware where FMA instructions are available (such as, Intel and ARM64). FMA intrinsics are generated for the java.lang.Math.fma(a, b, c) methods that calculate the value of ( a \* b + c ) expressions. |
| -XX:+UseRTMDeopt | Autotunes RTM locking depending on the abort ratio. This ratio is specified by the -XX:RTMAbortRatio option. If the number of aborted transactions exceeds the abort ratio, then the method containing the lock is deoptimized and recompiled with all locks as normal locks. This option is disabled by default. The -XX:+UseRTMLocking option must be enabled. |
| -XX:+UseRTMLocking | Generates Restricted Transactional Memory (RTM) locking code for all inflated locks, with the normal locking mechanism as the fallback handler. This option is disabled by default. Options related to RTM are available only on x86 CPUs that support Transactional Synchronization Extensions (TSX).  RTM is part of Intel's TSX, which is an x86 instruction set extension and facilitates the creation of multithreaded applications. RTM introduces the new instructions XBEGIN, XABORT, XEND, and XTEST. The XBEGIN and XEND instructions enclose a set of instructions to run as a transaction. If no conflict is found when running the transaction, then the memory and register modifications are committed together at the XEND instruction. The XABORT instruction can be used to explicitly abort a transaction and the XTEST instruction checks if a set of instructions is being run in a transaction.  A lock on a transaction is inflated when another thread tries to access the same transaction, thereby blocking the thread that didn't originally request access to the transaction. RTM requires that a fallback set of operations be specified in case a transaction aborts or fails. An RTM lock is a lock that has been delegated to the TSX's system.  RTM improves performance for highly contended locks with low conflict in a critical region (which is code that must not be accessed by more than one thread concurrently). RTM also improves the performance of coarse-grain locking, which typically doesn't perform well in multithreaded applications. (Coarse-grain locking is the strategy of holding locks for long periods to minimize the overhead of taking and releasing locks, while fine-grained locking is the strategy of trying to achieve maximum parallelism by locking only when necessary and unlocking as soon as possible.) Also, for lightly contended locks that are used by different threads, RTM can reduce false cache line sharing, also known as cache line ping-pong. This occurs when multiple threads from different processors are accessing different resources, but the resources share the same cache line. As a result, the processors repeatedly invalidate the cache lines of other processors, which forces them to read from main memory instead of their cache. |
| -XX:+UseSuperWord | Enables the transformation of scalar operations into superword operations. Superword is a vectorization optimization. This option is enabled by default. To disable the transformation of scalar operations into superword operations, specify -XX:-UseSuperWord. |

### Advanced Serviceability Options for Java

These java options provide the ability to gather system information and perform extensive debugging.

|  |  |
| --- | --- |
| -XX:+DisableAttachMechanism | Disables the mechanism that lets tools attach to the JVM. By default, this option is disabled, meaning that the attach mechanism is enabled and you can use diagnostics and troubleshooting tools such as jcmd, jstack, jmap, and jinfo.  Note: The tools such as jcmd, jinfo, jmap, and jstack shipped with the JDK aren't supported when using the tools from one JDK version to troubleshoot a different JDK version. |
| -XX:+ExtendedDTraceProbes | Linux and macOS: Enables additional dtrace tool probes that affect the performance. By default, this option is disabled and dtrace performs only standard probes. |
| -XX:+HeapDumpOnOutOfMemoryError | Enables the dumping of the Java heap to a file in the current directory by using the heap profiler (HPROF) when a java.lang.OutOfMemoryError exception is thrown. You can explicitly set the heap dump file path and name using the -XX:HeapDumpPath option. By default, this option is disabled and the heap isn't dumped when an OutOfMemoryError exception is thrown. |
| -XX:HeapDumpPath=path | Sets the path and file name for writing the heap dump provided by the heap profiler (HPROF) when the -XX:+HeapDumpOnOutOfMemoryError option is set. By default, the file is created in the current working directory, and it's named java\_pid<pid>.hprof where <pid> is the identifier of the process that caused the error. The following example shows how to set the default file explicitly (%p represents the current process identifier):  -XX:HeapDumpPath=./java\_pid%p.hprof  Linux and macOS: The following example shows how to set the heap dump file to /var/log/java/java\_heapdump.hprof:  -XX:HeapDumpPath=/var/log/java/java\_heapdump.hprof |
| -XX:LogFile=path | Sets the path and file name to where log data is written. By default, the file is created in the current working directory, and it's named hotspot.log.  Linux and macOS: The following example shows how to set the log file to /var/log/java/hotspot.log:  -XX:LogFile=/var/log/java/hotspot.log |
| -XX:+PrintClassHistogram | Enables printing of a class instance histogram after one of the following events:  Linux and macOS: Control+Break  Windows: Control+C (SIGTERM)  By default, this option is disabled.  Setting this option is equivalent to running the jmap -histo command, or the jcmd pid GC.class\_histogram command, where pid is the current Java process identifier. |
| -XX:+PrintConcurrentLocks | Enables printing of java.util.concurrent locks after one of the following events:  Linux and macOS: Control+Break  Windows: Control+C (SIGTERM)  By default, this option is disabled.  Setting this option is equivalent to running the jstack -l command or the jcmd pid Thread.print -l command, where pid is the current Java process identifier. |
| -XX:+PrintFlagsRanges | Prints the range specified and allows automatic testing of the values. See Validate Java Virtual Machine Flag Arguments. |
| -XX:+PerfDataSaveToFile | If enabled, saves jstat binary data when the Java application exits. This binary data is saved in a file named hsperfdata\_pid, where pid is the process identifier of the Java application that you ran. Use the jstat command to display the performance data contained in this file as follows:  jstat -class file:///path/hsperfdata\_pid  jstat -gc file:///path/hsperfdata\_pid |
| -XX:+UsePerfData | Enables the perfdata feature. This option is enabled by default to allow JVM monitoring and performance testing. Disabling it suppresses the creation of the hsperfdata\_userid directories. To disable the perfdata feature, specify -XX:-UsePerfData. |

### Advanced Garbage Collection Options for Java

These java options control how garbage collection (GC) is performed by the Java HotSpot VM.

|  |  |
| --- | --- |
| -XX:+AggressiveHeap | Enables Java heap optimization. This sets various parameters to be optimal for long-running jobs with intensive memory allocation, based on the configuration of the computer (RAM and CPU). By default, the option is disabled and the heap sizes are configured less aggressively. |
| -XX:+AlwaysPreTouch | Requests the VM to touch every page on the Java heap after requesting it from the operating system and before handing memory out to the application. By default, this option is disabled and all pages are committed as the application uses the heap space. |
| -XX:ConcGCThreads=threads | Sets the number of threads used for concurrent GC. Sets threads to approximately 1/4 of the number of parallel garbage collection threads. The default value depends on the number of CPUs available to the JVM.  For example, to set the number of threads for concurrent GC to 2, specify the following option:  -XX:ConcGCThreads=2 |
| -XX:+DisableExplicitGC | Enables the option that disables processing of calls to the System.gc() method. This option is disabled by default, meaning that calls to System.gc() are processed. If processing of calls to System.gc() is disabled, then the JVM still performs GC when necessary. |
| -XX:+ExplicitGCInvokesConcurrent | Enables invoking of concurrent GC by using the System.gc() request. This option is disabled by default and can be enabled only with the -XX:+UseG1GC option. |
| -XX:G1AdaptiveIHOPNumInitialSamples=number | When -XX:UseAdaptiveIHOP is enabled, this option sets the number of completed marking cycles used to gather samples until G1 adaptively determines the optimum value of -XX:InitiatingHeapOccupancyPercent. Before, G1 uses the value of -XX:InitiatingHeapOccupancyPercent directly for this purpose. The default value is 3. |
| -XX:G1HeapRegionSize=size | Sets the size of the regions into which the Java heap is subdivided when using the garbage-first (G1) collector. The value is a power of 2 and can range from 1 MB to 32 MB. The default region size is determined ergonomically based on the heap size with a goal of approximately 2048 regions.  The following example sets the size of the subdivisions to 16 MB:  -XX:G1HeapRegionSize=16m |
| -XX:G1HeapWastePercent=percent | Sets the percentage of heap that you're willing to waste. The Java HotSpot VM doesn't initiate the mixed garbage collection cycle when the reclaimable percentage is less than the heap waste percentage. The default is 5 percent. |
| -XX:G1MaxNewSizePercent=percent | Sets the percentage of the heap size to use as the maximum for the young generation size. The default value is 60 percent of your Java heap.  This is an experimental flag. This setting replaces the -XX:DefaultMaxNewGenPercent setting. |
| -XX:G1MixedGCCountTarget=number | Sets the target number of mixed garbage collections after a marking cycle to collect old regions with at most G1MixedGCLIveThresholdPercent live data. The default is 8 mixed garbage collections. The goal for mixed collections is to be within this target number. |
| -XX:G1MixedGCLiveThresholdPercent=percent | Sets the occupancy threshold for an old region to be included in a mixed garbage collection cycle. The default occupancy is 85 percent.  This is an experimental flag. This setting replaces the -XX:G1OldCSetRegionLiveThresholdPercent setting. |
| -XX:G1NewSizePercent=percent | Sets the percentage of the heap to use as the minimum for the young generation size. The default value is 5 percent of your Java heap.  This is an experimental flag. This setting replaces the -XX:DefaultMinNewGenPercent setting. |
| -XX:G1OldCSetRegionThresholdPercent=percent | Sets an upper limit on the number of old regions to be collected during a mixed garbage collection cycle. The default is 10 percent of the Java heap. |
| -XX:G1ReservePercent=percent | Sets the percentage of the heap (0 to 50) that's reserved as a false ceiling to reduce the possibility of promotion failure for the G1 collector. When you increase or decrease the percentage, ensure that you adjust the total Java heap by the same amount. By default, this option is set to 10%.  The following example sets the reserved heap to 20%:  -XX:G1ReservePercent=20 |
| -XX:+G1UseAdaptiveIHOP | Controls adaptive calculation of the old generation occupancy to start background work preparing for an old generation collection. If enabled, G1 uses -XX:InitiatingHeapOccupancyPercent for the first few times as specified by the value of -XX:G1AdaptiveIHOPNumInitialSamples, and after that adaptively calculates a new optimum value for the initiating occupancy automatically. Otherwise, the old generation collection process always starts at the old generation occupancy determined by -XX:InitiatingHeapOccupancyPercent.  The default is enabled. |
| -XX:InitialHeapSize=size | Sets the initial size (in bytes) of the memory allocation pool. This value must be either 0, or a multiple of 1024 and greater than 1 MB. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value is selected at run time based on the system configuration.  The following examples show how to set the size of allocated memory to 6 MB using various units:  -XX:InitialHeapSize=6291456  -XX:InitialHeapSize=6144k  -XX:InitialHeapSize=6m  If you set this option to 0, then the initial size is set as the sum of the sizes allocated for the old generation and the young generation. The size of the heap for the young generation can be set using the -XX:NewSize option. |
| -XX:InitialRAMPercentage=percent | Sets the initial amount of memory that the JVM will use for the Java heap before applying ergonomics heuristics as a percentage of the maximum amount determined as described in the -XX:MaxRAM option. The default value is 1.5625 percent.  The following example shows how to set the percentage of the initial amount of memory used for the Java heap:  -XX:InitialRAMPercentage=5 |
| -XX:InitialSurvivorRatio=ratio | Sets the initial survivor space ratio used by the throughput garbage collector (which is enabled by the -XX:+UseParallelGC option). Adaptive sizing is enabled by default with the throughput garbage collector by using the -XX:+UseParallelGC option, and the survivor space is resized according to the application behavior, starting with the initial value. If adaptive sizing is disabled (using the -XX:-UseAdaptiveSizePolicy option), then the -XX:SurvivorRatio option should be used to set the size of the survivor space for the entire execution of the application.  The following formula can be used to calculate the initial size of survivor space (S) based on the size of the young generation (Y), and the initial survivor space ratio (R):  S=Y/(R+2)  The 2 in the equation denotes two survivor spaces. The larger the value specified as the initial survivor space ratio, the smaller the initial survivor space size.  By default, the initial survivor space ratio is set to 8. If the default value for the young generation space size is used (2 MB), then the initial size of the survivor space is 0.2 MB.  The following example shows how to set the initial survivor space ratio to 4:  -XX:InitialSurvivorRatio=4 |
| -XX:InitiatingHeapOccupancyPercent=percent | Sets the percentage of the old generation occupancy (0 to 100) at which to start the first few concurrent marking cycles for the G1 garbage collector.  By default, the initiating value is set to 45%. A value of 0 implies nonstop concurrent GC cycles from the beginning until G1 adaptively sets this value.  See also the -XX:G1UseAdaptiveIHOP and -XX:G1AdaptiveIHOPNumInitialSamples options.  The following example shows how to set the initiating heap occupancy to 75%:  -XX:InitiatingHeapOccupancyPercent=75 |
| -XX:MaxGCPauseMillis=time | Sets a target for the maximum GC pause time (in milliseconds). This is a soft goal, and the JVM will make its best effort to achieve it. The specified value doesn't adapt to your heap size. By default, for G1 the maximum pause time target is 200 milliseconds. The other generational collectors do not use a pause time goal by default.  The following example shows how to set the maximum target pause time to 500 ms:  -XX:MaxGCPauseMillis=500 |
| -XX:MaxHeapSize=size | Sets the maximum size (in byes) of the memory allocation pool. This value must be a multiple of 1024 and greater than 2 MB. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value is selected at run time based on the system configuration. For server deployments, the options -XX:InitialHeapSize and -XX:MaxHeapSize are often set to the same value.  The following examples show how to set the maximum allowed size of allocated memory to 80 MB using various units:  -XX:MaxHeapSize=83886080  -XX:MaxHeapSize=81920k  -XX:MaxHeapSize=80m  The -XX:MaxHeapSize option is equivalent to -Xmx. |
| -XX:MaxHeapFreeRatio=percent | Sets the maximum allowed percentage of free heap space (0 to 100) after a GC event. If free heap space expands above this value, then the heap is shrunk. By default, this value is set to 70%.  Minimize the Java heap size by lowering the values of the parameters MaxHeapFreeRatio (default value is 70%) and MinHeapFreeRatio (default value is 40%) with the command-line options -XX:MaxHeapFreeRatio and -XX:MinHeapFreeRatio. Lowering MaxHeapFreeRatio to as low as 10% and MinHeapFreeRatio to 5% has successfully reduced the heap size without too much performance regression; however, results may vary greatly depending on your application. Try different values for these parameters until they're as low as possible yet still retain acceptable performance.  -XX:MaxHeapFreeRatio=10 -XX:MinHeapFreeRatio=5  Customers trying to keep the heap small should also add the option -XX:-ShrinkHeapInSteps. See Performance Tuning Examples for a description of using this option to keep the Java heap small by reducing the dynamic footprint for embedded applications. |
| -XX:MaxMetaspaceSize=size | Sets the maximum amount of native memory that can be allocated for class metadata. By default, the size isn't limited. The amount of metadata for an application depends on the application itself, other running applications, and the amount of memory available on the system.  The following example shows how to set the maximum class metadata size to 256 MB:  -XX:MaxMetaspaceSize=256m |
| -XX:MaxNewSize=size | Sets the maximum size (in bytes) of the heap for the young generation (nursery). The default value is set ergonomically. |
| -XX:MaxRAM=size | Sets the maximum amount of memory that the JVM may use for the Java heap before applying ergonomics heuristics. The default value is the maximum amount of available memory to the JVM process or 128 GB, whichever is lower.  The maximum amount of available memory to the JVM process is the minimum of the machine's physical memory and any constraints set by the environment (e.g. container).  Specifying this option disables automatic use of compressed oops if the combined result of this and other options influencing the maximum amount of memory is larger than the range of memory addressable by compressed oops. See -XX:UseCompressedOops for further information about compressed oops.  The following example shows how to set the maximum amount of available memory for sizing the Java heap to 2 GB:  -XX:MaxRAM=2G |
| -XX:MaxRAMPercentage=percent | Sets the maximum amount of memory that the JVM may use for the Java heap before applying ergonomics heuristics as a percentage of the maximum amount determined as described in the -XX:MaxRAM option. The default value is 25 percent.  Specifying this option disables automatic use of compressed oops if the combined result of this and other options influencing the maximum amount of memory is larger than the range of memory addressable by compressed oops. See -XX:UseCompressedOops for further information about compressed oops.  The following example shows how to set the percentage of the maximum amount of memory used for the Java heap:  -XX:MaxRAMPercentage=75 |
| -XX:MinRAMPercentage=percent | Sets the maximum amount of memory that the JVM may use for the Java heap before applying ergonomics heuristics as a percentage of the maximum amount determined as described in the -XX:MaxRAM option for small heaps. A small heap is a heap of approximately 125 MB. The default value is 50 percent.  The following example shows how to set the percentage of the maximum amount of memory used for the Java heap for small heaps:  -XX:MinRAMPercentage=75 |
| -XX:MaxTenuringThreshold=threshold | Sets the maximum tenuring threshold for use in adaptive GC sizing. The largest value is 15. The default value is 15 for the parallel (throughput) collector.  The following example shows how to set the maximum tenuring threshold to 10:  -XX:MaxTenuringThreshold=10 |
| -XX:MetaspaceSize=size | Sets the size of the allocated class metadata space that triggers a garbage collection the first time it's exceeded. This threshold for a garbage collection is increased or decreased depending on the amount of metadata used. The default size depends on the platform. |
| -XX:MinHeapFreeRatio=percent | Sets the minimum allowed percentage of free heap space (0 to 100) after a GC event. If free heap space falls below this value, then the heap is expanded. By default, this value is set to 40%.  Minimize Java heap size by lowering the values of the parameters MaxHeapFreeRatio (default value is 70%) and MinHeapFreeRatio (default value is 40%) with the command-line options -XX:MaxHeapFreeRatio and -XX:MinHeapFreeRatio. Lowering MaxHeapFreeRatio to as low as 10% and MinHeapFreeRatio to 5% has successfully reduced the heap size without too much performance regression; however, results may vary greatly depending on your application. Try different values for these parameters until they're as low as possible, yet still retain acceptable performance.  -XX:MaxHeapFreeRatio=10 -XX:MinHeapFreeRatio=5  Customers trying to keep the heap small should also add the option -XX:-ShrinkHeapInSteps. See Performance Tuning Examples for a description of using this option to keep the Java heap small by reducing the dynamic footprint for embedded applications. |
| -XX:MinHeapSize=size | Sets the minimum size (in bytes) of the memory allocation pool. This value must be either 0, or a multiple of 1024 and greater than 1 MB. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. The default value is selected at run time based on the system configuration.  The following examples show how to set the mimimum size of allocated memory to 6 MB using various units:  -XX:MinHeapSize=6291456  -XX:MinHeapSize=6144k  -XX:MinHeapSize=6m  If you set this option to 0, then the minimum size is set to the same value as the initial size. |
| -XX:NewRatio=ratio | Sets the ratio between young and old generation sizes. By default, this option is set to 2. The following example shows how to set the young-to-old ratio to 1:  -XX:NewRatio=1 |
| -XX:NewSize=size | Sets the initial size (in bytes) of the heap for the young generation (nursery). Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes.  The young generation region of the heap is used for new objects. GC is performed in this region more often than in other regions. If the size for the young generation is too low, then a large number of minor GCs are performed. If the size is too high, then only full GCs are performed, which can take a long time to complete. It is recommended that you keep the size for the young generation greater than 25% and less than 50% of the overall heap size.  The following examples show how to set the initial size of the young generation to 256 MB using various units:  -XX:NewSize=256m  -XX:NewSize=262144k  -XX:NewSize=268435456  The -XX:NewSize option is equivalent to -Xmn. |
| -XX:ParallelGCThreads=threads | Sets the number of the stop-the-world (STW) worker threads. The default value depends on the number of CPUs available to the JVM and the garbage collector selected.  For example, to set the number of threads for G1 GC to 2, specify the following option:  -XX:ParallelGCThreads=2 |
| -XX:+ParallelRefProcEnabled | Enables parallel reference processing. By default, this option is disabled. |
| -XX:+PrintAdaptiveSizePolicy | Enables printing of information about adaptive-generation sizing. By default, this option is disabled. |
| -XX:+ScavengeBeforeFullGC | Enables GC of the young generation before each full GC. This option is enabled by default. It is recommended that you don't disable it, because scavenging the young generation before a full GC can reduce the number of objects reachable from the old generation space into the young generation space. To disable GC of the young generation before each full GC, specify the option -XX:-ScavengeBeforeFullGC. |
| -XX:SoftRefLRUPolicyMSPerMB=time | Sets the amount of time (in milliseconds) a softly reachable object is kept active on the heap after the last time it was referenced. The default value is one second of lifetime per free megabyte in the heap. The -XX:SoftRefLRUPolicyMSPerMB option accepts integer values representing milliseconds per one megabyte of the current heap size (for Java HotSpot Client VM) or the maximum possible heap size (for Java HotSpot Server VM). This difference means that the Client VM tends to flush soft references rather than grow the heap, whereas the Server VM tends to grow the heap rather than flush soft references. In the latter case, the value of the -Xmx option has a significant effect on how quickly soft references are garbage collected.  The following example shows how to set the value to 2.5 seconds:  -XX:SoftRefLRUPolicyMSPerMB=2500 |
| -XX:-ShrinkHeapInSteps | Incrementally reduces the Java heap to the target size, specified by the option -XX:MaxHeapFreeRatio. This option is enabled by default. If disabled, then it immediately reduces the Java heap to the target size instead of requiring multiple garbage collection cycles. Disable this option if you want to minimize the Java heap size. You will likely encounter performance degradation when this option is disabled.  See Performance Tuning Examples for a description of using the MaxHeapFreeRatio option to keep the Java heap small by reducing the dynamic footprint for embedded applications. |
| -XX:StringDeduplicationAgeThreshold=threshold | Identifies String objects reaching the specified age that are considered candidates for deduplication. An object's age is a measure of how many times it has survived garbage collection. This is sometimes referred to as tenuring.  Note: String objects that are promoted to an old heap region before this age has been reached are always considered candidates for deduplication. The default value for this option is 3. See the -XX:+UseStringDeduplication option. |
| -XX:SurvivorRatio=ratio | Sets the ratio between eden space size and survivor space size. By default, this option is set to 8. The following example shows how to set the eden/survivor space ratio to 4:  -XX:SurvivorRatio=4 |
| -XX:TargetSurvivorRatio=percent | Sets the desired percentage of survivor space (0 to 100) used after young garbage collection. By default, this option is set to 50%.  The following example shows how to set the target survivor space ratio to 30%:  -XX:TargetSurvivorRatio=30 |
| -XX:TLABSize=size | Sets the initial size (in bytes) of a thread-local allocation buffer (TLAB). Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, or g or G to indicate gigabytes. If this option is set to 0, then the JVM selects the initial size automatically.  The following example shows how to set the initial TLAB size to 512 KB:  -XX:TLABSize=512k |
| -XX:+UseAdaptiveSizePolicy | Enables the use of adaptive generation sizing. This option is enabled by default. To disable adaptive generation sizing, specify -XX:-UseAdaptiveSizePolicy and set the size of the memory allocation pool explicitly. See the -XX:SurvivorRatio option. |
| -XX:+UseG1GC | Enables the use of the garbage-first (G1) garbage collector. It's a server-style garbage collector, targeted for multiprocessor machines with a large amount of RAM. This option meets GC pause time goals with high probability, while maintaining good throughput. The G1 collector is recommended for applications requiring large heaps (sizes of around 6 GB or larger) with limited GC latency requirements (a stable and predictable pause time below 0.5 seconds). By default, this option is enabled and G1 is used as the default garbage collector. |
| -XX:+UseGCOverheadLimit | Enables the use of a policy that limits the proportion of time spent by the JVM on GC before an OutOfMemoryError exception is thrown. This option is enabled, by default, and the parallel GC will throw an OutOfMemoryError if more than 98% of the total time is spent on garbage collection and less than 2% of the heap is recovered. When the heap is small, this feature can be used to prevent applications from running for long periods of time with little or no progress. To disable this option, specify the option -XX:-UseGCOverheadLimit. |
| -XX:+UseNUMA | Enables performance optimization of an application on a machine with nonuniform memory architecture (NUMA) by increasing the application's use of lower latency memory. By default, this option is disabled and no optimization for NUMA is made. The option is available only when the parallel garbage collector is used (-XX:+UseParallelGC). |
| -XX:+UseParallelGC | Enables the use of the parallel scavenge garbage collector (also known as the throughput collector) to improve the performance of your application by leveraging multiple processors.  By default, this option is disabled and the default collector is used. |
| -XX:+UseSerialGC | Enables the use of the serial garbage collector. This is generally the best choice for small and simple applications that don't require any special functionality from garbage collection. By default, this option is disabled and the default collector is used. |
| -XX:+UseSHM | Linux only: Enables the JVM to use shared memory to set up large pages.  See Large Pages for setting up large pages. |
| -XX:+UseStringDeduplication | Enables string deduplication. By default, this option is disabled. To use this option, you must enable the garbage-first (G1) garbage collector.  String deduplication reduces the memory footprint of String objects on the Java heap by taking advantage of the fact that many String objects are identical. Instead of each String object pointing to its own character array, identical String objects can point to and share the same character array. |
| -XX:+UseTLAB | Enables the use of thread-local allocation blocks (TLABs) in the young generation space. This option is enabled by default. To disable the use of TLABs, specify the option -XX:-UseTLAB. |
| -XX:+UseZGC | Enables the use of the Z garbage collector (ZGC). This is a low latency garbage collector, providing max pause times of a few milliseconds, at some throughput cost. Pause times are independent of what heap size is used. Supports heap sizes from 8MB to 16TB. |
| -XX:ZAllocationSpikeTolerance=factor | Sets the allocation spike tolerance for ZGC. By default, this option is set to 2.0. This factor describes the level of allocation spikes to expect. For example, using a factor of 3.0 means the current allocation rate can be expected to triple at any time. |
| -XX:ZCollectionInterval=seconds | Sets the maximum interval (in seconds) between two GC cycles when using ZGC. By default, this option is set to 0 (disabled). |
| -XX:ZFragmentationLimit=percent | Sets the maximum acceptable heap fragmentation (in percent) for ZGC. By default, this option is set to 25. Using a lower value will cause the heap to be compacted more aggressively, to reclaim more memory at the cost of using more CPU time. |
| -XX:+ZProactive | Enables proactive GC cycles when using ZGC. By default, this option is enabled. ZGC will start a proactive GC cycle if doing so is expected to have minimal impact on the running application. This is useful if the application is mostly idle or allocates very few objects, but you still want to keep the heap size down and allow reference processing to happen even when there are a lot of free space on the heap. |
| -XX:+ZUncommit | Enables uncommitting of unused heap memory when using ZGC. By default, this option is enabled. Uncommitting unused heap memory will lower the memory footprint of the JVM, and make that memory available for other processes to use. |
| -XX:ZUncommitDelay=seconds | Sets the amount of time (in seconds) that heap memory must have been unused before being uncommitted. By default, this option is set to 300 (5 minutes). Committing and uncommitting memory are relatively expensive operations. Using a lower value will cause heap memory to be uncommitted earlier, at the risk of soon having to commit it again. |

### Using the JDK\_JAVA\_OPTIONS Launcher Environment Variable

* + - 1. JDK\_JAVA\_OPTIONS prepends its content to the options parsed from the command line.
      2. The content of the JDK\_JAVA\_OPTIONS environment variable is a list of arguments separated by white-space characters (as determined by isspace()).
      3. These are prepended to the command line arguments passed to java launcher.
      4. The encoding requirement for the environment variable is the same as the java command line on the system.
      5. JDK\_JAVA\_OPTIONS environment variable content is treated in the same manner as that specified in the command line.
      6. Single (') or double (") quotes can be used to enclose arguments that contain whitespace characters.
         1. All content between the open quote and the first matching close quote are preserved by simply removing the pair of quotes.
         2. In case a matching quote is not found, the launcher will abort with an error message.
      7. @-files are supported as they are specified in the command line.
         1. However, as in @-files, use of a wildcard is not supported.
      8. In order to mitigate potential misuse of JDK\_JAVA\_OPTIONS behavior
         1. options that specify the main class (such as -jar) or cause the java launcher to exit without executing the main class (such as -h) are disallowed in the environment variable.
         2. If any of these options appear in the environment variable, the launcher will abort with an error message.
      9. When JDK\_JAVA\_OPTIONS is set, the launcher prints a message to stderr as a reminder.
      10. Example:

$ export JDK\_JAVA\_OPTIONS='-g @file1 -Dprop=value @file2 -Dws.prop="white spaces"'

$ java -Xint @file3

is equivalent to the command line:

java -g @file1 -Dprop=value @file2 -Dws.prop="white spaces" -Xint @file3

### Enable Logging with the JVM Unified Logging Framework

* + - 1. You use the -Xlog option to configure or enable logging with the Java Virtual Machine (JVM) unified logging framework.
      2. Description

The Java Virtual Machine (JVM) unified logging framework provides a common logging system for all components of the JVM. GC logging for the JVM has been changed to use the new logging framework. The mapping of old GC flags to the corresponding new Xlog configuration is described in Convert GC Logging Flags to Xlog. In addition, runtime logging has also been changed to use the JVM unified logging framework. The mapping of legacy runtime logging flags to the corresponding new Xlog configuration is described in Convert Runtime Logging Flags to Xlog.

The following provides quick reference to the -Xlog command and syntax for options:

-Xlog

Enables JVM logging on an info level.

-Xlog:help

Prints -Xlog usage syntax and available tags, levels, and decorators along with example command lines with explanations.

-Xlog:disable

Turns off all logging and clears all configuration of the logging framework including the default configuration for warnings and errors.

-Xlog[:option]

Applies multiple arguments in the order that they appear on the command line. Multiple -Xlog arguments for the same output override each other in their given order.

The option is set as:

[tag-selection][:[output][:[decorators][:output-options]]]

Omitting the tag-selection defaults to a tag-set of all and a level of info.

tag[+...] all

The all tag is a meta tag consisting of all tag-sets available. The asterisk \* in a tag set definition denotes a wildcard tag match. Matching with a wildcard selects all tag sets that contain at least the specified tags. Without the wildcard, only exact matches of the specified tag sets are selected.

output-options is

filecount=file-count filesize=file size with optional K, M or G suffix

* + - 1. Synopsis

-Xlog[:[what][:[output][:[decorators][:output-options[,...]]]]]

what

Specifies a combination of tags and levels of the form tag1[+tag2...][\*][=level][,...]. Unless the wildcard (\*) is specified, only log messages tagged with exactly the tags specified are matched. See -Xlog Tags and Levels.

output

Sets the type of output. Omitting the output type defaults to stdout. See -Xlog Output.

decorators

Configures the output to use a custom set of decorators. Omitting decorators defaults to uptime, level, and tags. See Decorations.

output-options

Sets the -Xlog logging output options.

* + - 1. Xlog Usage Examples

### Large Pages

You use large pages, also known as huge pages, as memory pages that are significantly larger than the standard memory page size (which varies depending on the processor and operating system). Large pages optimize processor Translation-Lookaside Buffers.

A Translation-Lookaside Buffer (TLB) is a page translation cache that holds the most-recently used virtual-to-physical address translations. A TLB is a scarce system resource. A TLB miss can be costly because the processor must then read from the hierarchical page table, which may require multiple memory accesses. By using a larger memory page size, a single TLB entry can represent a larger memory range. This results in less pressure on a TLB, and memory-intensive applications may have better performance.

However, large pages page memory can negatively affect system performance. For example, when a large mount of memory is pinned by an application, it may create a shortage of regular memory and cause excessive paging in other applications and slow down the entire system. Also, a system that has been up for a long time could produce excessive fragmentation, which could make it impossible to reserve enough large page memory. When this happens, either the OS or JVM reverts to using regular pages.

Linux and Windows support large pages.

Large Pages Support for Linux

The 2.6 kernel supports large pages. Some vendors have backported the code to their 2.4-based releases. To check if your system can support large page memory, try the following:

# cat /proc/meminfo | grep Huge

HugePages\_Total: 0

HugePages\_Free: 0

Hugepagesize: 2048 kB

If the output shows the three "Huge" variables, then your system can support large page memory but it needs to be configured. If the command prints nothing, then your system doesn't support large pages. To configure the system to use large page memory, login as root, and then follow these steps:

If you're using the option -XX:+UseSHM (instead of -XX:+UseHugeTLBFS), then increase the SHMMAX value. It must be larger than the Java heap size. On a system with 4 GB of physical RAM (or less), the following makes all the memory sharable:

# echo 4294967295 > /proc/sys/kernel/shmmax

If you're using the option -XX:+UseSHM or -XX:+UseHugeTLBFS, then specify the number of large pages. In the following example, 3 GB of a 4 GB system are reserved for large pages (assuming a large page size of 2048kB, then 3 GB = 3 \* 1024 MB = 3072 MB = 3072 \* 1024 kB = 3145728 kB and 3145728 kB / 2048 kB = 1536):

# echo 1536 > /proc/sys/vm/nr\_hugepages

Note: The values contained in /proc resets after you reboot your system, so may want to set them in an initialization script (for example, rc.local or sysctl.conf).

If you configure (or resize) the OS kernel parameters /proc/sys/kernel/shmmax or /proc/sys/vm/nr\_hugepages, Java processes may allocate large pages for areas in addition to the Java heap. These steps can allocate large pages for the following areas:

Java heap

Code cache

The marking bitmap data structure for the parallel GC

Consequently, if you configure the nr\_hugepages parameter to the size of the Java heap, then the JVM can fail in allocating the code cache areas on large pages because these areas are quite large in size.

### Application Class Data Sharing

### Example

### Deprecated Java Options

* + - 1. jdk16
      2. jdk15
      3. jdk14
      4. jdk13
      5. jdk12
      6. jdk11
      7. jdk10
      8. jdk9

### Obsolete Java Options

* + - 1. jdk16
      2. jdk15
      3. jdk14
      4. jdk13
      5. jdk12
      6. jdk11
      7. jdk10
      8. jdk9

### Remove Java Options

* + - 1. jdk16
      2. jdk15
      3. jdk14
      4. jdk13
      5. jdk12
      6. jdk11
      7. jdk10
      8. jdk9

## jar

### Description

The jar command is a general-purpose archiving and compression tool, based on the ZIP and ZLIB compression formats.

Initially, the jar command was designed to package Java applets (not supported since JDK 11) or applications;

However, beginning with JDK 9, users can use the jar command to create modular JARs.

For transportation and deployment, it's usually more convenient to package modules as modular JARs.

The syntax for the jar command resembles the syntax for the tar command.

It has several main operation modes, defined by one of the mandatory operation arguments.

Other arguments are either options that modify the behavior of the operation or are required to perform the operation.

When modules or the components of an application (files, images and sounds) are combined into a single archive, they can be downloaded by a Java agent (such as a browser) in a single HTTP transaction, rather than requiring a new connection for each piece. This dramatically improves download times.

The jar command also compresses files, which further improves download time.

The jar command also enables individual entries in a file to be signed so that their origin can be authenticated.

A JAR file can be used as a class path entry, whether or not it's compressed.

An archive becomes a modular JAR when you include a module descriptor, module-info.class, in the root of the given directories or in the root of the .jar archive.

All mandatory or optional arguments for long options are also mandatory or optional for any corresponding short options.

### Synopsis

jar [OPTION ...] [ [--release VERSION] [-C dir] files] ...

### Options

* + - 1. Main Operation Modes

When using the jar command, you must specify the operation for it to perform.

You specify the operation mode for the jar command by including the appropriate operation arguments described in this section.

You can mix an operation argument with other one-letter options.

Generally the operation argument is the first argument specified on the command line.

Options

-c or --create

Creates the archive.

-i=FILE or --generate-index=FILE

Generates index information for the specified JAR file.

-t or --list

Lists the table of contents for the archive.

-u or --update

Updates an existing JAR file.

-x or --extract

Extracts the named (or all) files from the archive.

-d or --describe-module

Prints the module descriptor or automatic module name.

* + - 1. Operation Modifiers Valid in Any mode

You can use the following options to customize the actions of any operation mode included in the jar command.

Options

-C DIR

Changes the specified directory and includes the files specified at the end of the command line.

-f=FILE or --file=FILE

Specifies the archive file name.

--release VERSION

Creates a multirelease JAR file.

Places all files specified after the option into a versioned directory of the JAR file named META-INF/versions/VERSION/, where VERSION must be must be a positive integer whose value is 9 or greater.

At run time, where more than one version of a class exists in the JAR, the JDK will use the first one it finds, searching initially in the directory tree whose VERSION number matches the JDK's major version number.

It will then look in directories with successively lower VERSION numbers, and finally look in the root of the JAR.

-v or --verbose

Sends or prints verbose output to standard output.

* + - 1. Operation Modifiers Valid Only in Create and Update Modes

You can use the following options to customize the actions of the create and the update main operation modes

Options

-e=CLASSNAME or --main-class=CLASSNAME

Specifies the application entry point for standalone applications bundled into a modular or executable modular JAR file.

-m=FILE or --manifest=FILE

Includes the manifest information from the given manifest file.

-M or --no-manifest

Doesn't create a manifest file for the entries.

--module-version=VERSION

Specifies the module version, when creating or updating a modular JAR file, or updating a non-modular JAR file.

--hash-modules=PATTERN

Computes and records the hashes of modules matched by the given pattern and that depend upon directly or indirectly on a modular JAR file being created or a non-modular JAR file being updated.

-p or --module-path

Specifies the location of module dependence for generating the hash.

@file

Reads jar options and file names from a text file.

* + - 1. Operation Modifiers Valid Only in Create, Update, and Generate-index Modes

You can use the following options to customize the actions of the create (-c or --create) the update (-u or --update ) and the generate-index (-i or --generate-index=FILE) main operation modes:

Options

-0 or --no-compress

Stores without using ZIP compression.

* + - 1. Other Options

The following options are recognized by the jar command and not used with operation modes:

Options

-h or --help[:compat]

Displays the command-line help for the jar command or optionally the compatibility help.

--help-extra

Displays help on extra options.

--version

Prints the program version.

### Example

* + - 1. Create an archive, classes.jar, that contains two class files, Foo.class and Bar.class.

jar --create --file classes.jar Foo.class Bar.class

* + - 1. Create an archive, classes.jar, by using an existing manifest, mymanifest, that contains all of the files in the directory foo/.

jar --create --file classes.jar --manifest mymanifest -C foo/

* + - 1. Create a modular JAR archive,foo.jar, where the module descriptor is located in classes/module-info.class.

jar --create --file foo.jar --main-class com.foo.Main --module-version 1.0 -C foo/classes resources

* + - 1. Update an existing non-modular JAR, foo.jar, to a modular JAR file.

jar --update --file foo.jar --main-class com.foo.Main --module-version 1.0 -C foo/module-info.class

* + - 1. Create a versioned or multi-release JAR, foo.jar, that places the files in the classes directory at the root of the JAR, and the files in the classes-10 directory in the META-INF/versions/10 directory of the JAR.
         1. In this example, the classes/com/foo directory contains two classes, com.foo.Hello (the entry point class) and com.foo.NameProvider, both compiled for JDK 8.
         2. The classes-10/com/foo directory contains a different version of the com.foo.NameProvider class, this one containing JDK 10 specific code and compiled for JDK 10.
         3. Given this setup, create a multirelease JAR file foo.jar by running the following command from the directory containing the directories classes and classes-10.

jar --create --file foo.jar --main-class com.foo.Hello -C classes . --release 10 -C classes-10 .

The JAR file foo.jar now contains:

% jar -tf foo.jar

META-INF/

META-INF/MANIFEST.MF

com/

com/foo/

com/foo/Hello.class

com/foo/NameProvider.class

META-INF/versions/10/com/

META-INF/versions/10/com/foo/

META-INF/versions/10/com/foo/NameProvider.class

As well as other information, the file META-INF/MANIFEST.MF, will contain the following lines to indicate that this is a multirelease JAR file with an entry point of com.foo.Hello.

...

Main-Class: com.foo.Hello

Multi-Release: true

Assuming that the com.foo.Hello class calls a method on the com.foo.NameProvider class, running the program using JDK 10 will ensure that the com.foo.NameProvider class is the one in META-INF/versions/10/com/foo/. Running the program using JDK 8 will ensure that the com.foo.NameProvider class is the one at the root of the JAR, in com/foo.

Create an archive, my.jar, by reading options and lists of class files from the file classes.list.

NOTE:To shorten or simplify the jar command, you can specify arguments in a separate text file and pass it to the jar command with the at sign (@) as a prefix.

jar --create --file my.jar @classes.list

## javap

### Description

* + - 1. The javap command disassembles one or more class files.
      2. The output depends on the options used.
      3. When no options are used, the javap command prints the protected and public fields, and methods of the classes passed to it.
      4. The javap command isn't multirelease JAR aware.
      5. Using the class path form of the command results in viewing the base entry in all JAR files, multirelease or not.
      6. Using the URL form, you can use the URL form of an argument to specify a specific version of a class to be disassembled.
      7. The javap command prints its output to stdout.
      8. Note:

In tools that support -- style options, the GNU-style options can use the equal sign (=) instead of a white space to separate the name of an option from its value.

### Synopsis

javap [options] classes...

options

Specifies the command-line options. See Options for javap.

classes

Specifies one or more classes separated by spaces to be processed for annotations.

You can specify a class that can be found in the class path by its file name, URL, or by its fully qualified class name.

Examples:

path/to/MyClass.class

jar:file:///path/to/MyJar.jar!/mypkg/MyClass.class

java.lang.Object

### Options for javap

|  |  |
| --- | --- |
| -verbose or -v | Prints additional information about the selected class. |
| -l | Prints line and local variable tables. |
| -public | Shows only public classes and members. |
| -protected | Shows only protected and public classes and members. |
| -package | Shows package/protected/public classes and members (default). |
| -private  -p | Shows all classes and members. |
| -c | Prints disassembled code,  for example, the instructions that comprise the Java bytecodes, for each of the methods in the class. |
| -s | Prints internal type signatures. |
| -sysinfo | Shows system information (path, size, date, SHA-256 hash) of the class being processed. |
| -constants | Shows static final constants. |
| --module module  -m module | Specifies the module containing classes to be disassembled. |
| --module-path path | Specifies where to find application modules. |
| --system jdk | Specifies where to find system modules. |
| --class-path path  -classpath path  -cp path | Specifies the path that the javap command uses to find user class files.  It overrides the default or the CLASSPATH environment variable when it's set. |
| -bootclasspath path | Overrides the location of bootstrap class files. |
| --multi-release version | Specifies the version to select in multi-release JAR files. |
| -Joption | Passes the specified option to the JVM.  For example:  javap -J-version  javap -J-Djava.security.manager -J-Djava.security.policy=MyPolicy MyClassName  See Overview of Java Options in java. |
| -version | Prints release information. |
| --help  -help  -h  -? | Prints a help message for the javap command. |

### Example

Compile the following HelloWorldFrame class:

import java.awt.Graphics;  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
  
public class HelloWorldFrame extends JFrame {  
  
 String message = "Hello World!";  
  
 public HelloWorldFrame() {  
 setContentPane(new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 g.drawString(message, 15, 30);  
 }  
 });  
 setSize(100, 100);  
 }  
   
 public static void main(String[] args) {  
 HelloWorldFrame frame = new HelloWorldFrame();  
 frame.setVisible(true);  
 }  
}

The output from the javap HelloWorldFrame.class command yields the following:

> Compiled from "HelloWorldFrame.java"

public class HelloWorldFrame extends javax.swing.JFrame {  
 java.lang.String message;  
 public HelloWorldFrame();  
 public static void main(java.lang.String[]);  
}

The output from the javap -c HelloWorldFrame.class command yields the following:

> Compiled from "HelloWorldFrame.java"

public class HelloWorldFrame extends javax.swing.JFrame {

java.lang.String message;

public HelloWorldFrame();

Code:

0: aload\_0

1: invokespecial #1 // Method javax/swing/JFrame."<init>":()V

4: aload\_0

5: ldc #2 // String Hello World!

7: putfield #3 // Field message:Ljava/lang/String;

10: aload\_0

11: new #4 // class HelloWorldFrame$1

14: dup

15: aload\_0

16: invokespecial #5 // Method HelloWorldFrame$1."<init>":(LHelloWorldFrame;)V

19: invokevirtual #6 // Method setContentPane:(Ljava/awt/Container;)V

22: aload\_0

23: bipush 100

25: bipush 100

27: invokevirtual #7 // Method setSize:(II)V

30: return

public static void main(java.lang.String[]);

Code:

0: new #8 // class HelloWorldFrame

3: dup

4: invokespecial #9 // Method "<init>":()V

7: astore\_1

8: aload\_1

9: iconst\_1

10: invokevirtual #10 // Method setVisible:(Z)V

13: return

}

# Other

## jaotc

### Description

* + - 1. The jaotc command is Java Ahead-Of-Time(AOT) static compiler which produces native code in the form of a shared library for the Java methods in specified Java class file.
      2. The Java Virtual Machine can load these AOT libraries and use native code from them when corresponding Java motheds are called.
      3. By use jaotc, there is no need to wait for the JIT compiler to generate (By compiling bytecode) the fast native code for these Java methods.
      4. The code is alread generated by jaotc and ready to be immediately used.
      5. For the same reason, these is no need to execute these methods in the Interpreter because fast compiled native code can be executed instead
      6. The jaotc command is experimental. JEP 295: Ahead-of-Time Compilation

### Synopsis

jaotc [options] [name|list]

options

Command-line options separated by space. see 1.3 Options

name

The Java class or jar file from which Java methods will be compiled

list

Colon separated list of class names,modules,jar files or directories which contain class file

### Options

--output file

Output file name. Default name is "unnamed.so".

--class-name class-names

List of Java classes to compile.

--jar jar-files

List of JAR files to compile.

--module modules

List of Java modules to compile.

--directory dirs

List of directories to search for files to compile.

--search-path dirs

List of directories to search for specified files.

--compile-commands file

Name of the file containing the compile commands:

exclude

Excludes compilation of specified methods.

compileOnly

Compiles only specified methods.

Regular expressions are used to specify classes and methods. For example:

exclude sun.util.resources..\*.TimeZoneNames\_.\*.getContents\(\)\[\[Ljava/lang/Object;

exclude sun.security.ssl.\*

compileOnly java.lang.String.\*

--compile-for-tiered

Generates profiling code for tiered compilation.

By default, profiling code is not generated (could be changed in a future).

--compile-with-assertions

Generates code with java assertions. By default, assertions code is not generated.

--compile-threads number

Sets the number of compilation threads used. The default value is min(16, available\_cpus).

--ignore-errors

Ignores all exceptions thrown during class loading.

By default, the tool will exit compilation if class loading throws an exception.

--exit-on-error

Exits on compilation errors.

By default, failed compilation is skipped and compilation of other methods continues.

--info

Prints information about compilation phases.

--verbose

Prints more details about compilation phases.

--debug

Prints comprehensive details.

--help or -h or -?

Prints a summary of standard options and exits the tool.

--version

Prints version information.

-Jflag

Provides a flag to pass to the runtime system.

To pass more than one flag, provide an instance of this option for each flag or flag argument needed.

### Example

* + - 1. Use the jaotc tool to execute AOT compilation.

jaotc --output libHelloWorld.so HelloWorld.class

* + - 1. Specify a generated AOT library during application execution:

java -XX:+UnlockExperimentalVMOptions -XX:AOTLibrary=./libHelloWorld.so HelloWorld

### What is Ahead of Time Compilation:

https://openjdk.java.net/jeps/295

https://www.baeldung.com/ahead-of-time-compilation

## jarsigner

### Description

* + - 1. 、The jarsigner tool has two purposes:

To sign Java Archive (JAR) files.

To verify the signatures and integrity of signed JAR files.

* + - 1. 、The JAR feature enables the packaging of class files, images, sounds, and other digital data in a single file for faster and easier distribution.A tool named jar enables developers to produce JAR files.
      2. 、Technically, any ZIP file can also be considered a JAR file, although when created by the jar command or processed by the jarsigner command, JAR files also contain a META-INF/MANIFEST.MF file.A digital signature is a string of bits that is computed from some data (the data being signed) and the private key of an entity (a person, company, and so on).
      3. 、Similar to a handwritten signature, a digital signature has many useful characteristics:

Its authenticity can be verified by a computation that uses the public key corresponding to the private key used to generate the signature.

It can't be forged, assuming the private key is kept secret.

It is a function of the data signed and thus can't be claimed to be the signature for other data as well.

The signed data can't be changed. If the data is changed, then the signature can't be verified as authentic.

* + - 1. 、To generate an entity's signature for a file, the entity must first have a public/private key pair associated with it and one or more certificates that authenticate its public key.
      2. 、A certificate is a digitally signed statement from one entity that says that the public key of another entity has a particular value.
      3. 、The jarsigner command uses key and certificate information from a keystore to generate digital signatures for JAR files. A keystore is a database of private keys and their associated X.509 certificate chains that authenticate the corresponding public keys. The keytool command is used to create and administer keystores.
      4. 、The jarsigner command uses an entity's private key to generate a signature. The signed JAR file contains, among other things, a copy of the certificate from the keystore for the public key corresponding to the private key used to sign the file. The jarsigner command can verify the digital signature of the signed JAR file using the certificate inside it (in its signature block file).
      5. 、The jarsigner command can generate signatures that include a time stamp that enables a systems or deployer to check whether the JAR file was signed while the signing certificate was still valid.
      6. 、In addition, APIs allow applications to obtain the timestamp information.
      7. 、At this time, the jarsigner command can only sign JAR files created by the jar command or zip files. JAR files are the same as zip files, except they also have a META-INF/MANIFEST.MF file. A META-INF/MANIFEST.MF file is created when the jarsigner command signs a zip file.
      8. 、The default jarsigner command behavior is to sign a JAR or zip file. Use the -verify option to verify a signed JAR file.
      9. 、The jarsigner command also attempts to validate the signer's certificate after signing or verifying. During validation, it checks the revocation status of each certificate in the signer's certificate chain when the -revCheck option is specified. If there is a validation error or any other problem, the command generates warning messages. If you specify the -strict option, then the command treats severe warnings as errors. See Errors and Warnings.

### Synopsis

jarsigner [options] jar-file alias

jarsigner -verify [options] jar-file [alias ...]

options

The command-line options. See Options for jarsigner.

-verify

The -verify option can take zero or more keystore alias names after the JAR file name. When the -verify option is specified, the jarsigner command checks that the certificate used to verify each signed entry in the JAR file matches one of the keystore aliases. The aliases are defined in the keystore specified by -keystore or the default keystore.

If you also specify the -strict option, and the jarsigner command detects severe warnings, the message, "jar verified, with signer errors" is displayed.

jar-file

The JAR file to be signed.

If you also specified the -strict option, and the jarsigner command detected severe warnings, the message, "jar signed, with signer errors" is displayed.

alias

The aliases are defined in the keystore specified by -keystore or the default keystore.

### Options

* + - 1. 、Be aware of the following standards:
         1. All option names are preceded by a hyphen sign (-)
         2. The options can be provided in any order.
         3. Items that are in italics or underlined (option values) represent the actual values that must be supplied.
         4. The -storepass, -keypass, -sigfile, -sigalg, -digestalg, -signedjar, and TSA-related options are only relevant when signing a JAR file; they aren't relevant when verifying a signed JAR file. The -keystore option is relevant for signing and verifying a JAR file. In addition, aliases are specified when signing and verifying a JAR file.
      2. 、Options

-keystore url

Specifies the URL that tells the keystore location. This defaults to the file .keystore in the user's home directory, as determined by the user.home system property.

A keystore is required when signing. You must explicitly specify a keystore when the default keystore doesn't exist or if you want to use one other than the default.

A keystore isn't required when verifying, but if one is specified or the default exists and the -verbose option was also specified, then additional information is output regarding whether or not any of the certificates used to verify the JAR file are contained in that keystore.

The -keystore argument can be a file name and path specification rather than a URL, in which case it is treated the same as a file: URL, for example, the following are equivalent:

-keystore filePathAndName

-keystore file:filePathAndName

If the Sun PKCS #11 provider was configured in the java.security security properties file (located in the JDK's $JAVA\_HOME/conf/security directory), then the keytool and jarsigner tools can operate on the PKCS #11 token by specifying these options:

-keystore NONE -storetype PKCS11

For example, the following command lists the contents of the configured PKCS#11 token:

keytool -keystore NONE -storetype PKCS11 -list

-storepass [:env | :file] argument

Specifies the password that is required to access the keystore. This is only needed when signing (not verifying) a JAR file. In that case, if a -storepass option isn't provided at the command line, then the user is prompted for the password.

If the modifier env or file isn't specified, then the password has the value argument. Otherwise, the password is retrieved as follows:

env: Retrieve the password from the environment variable named argument.

file: Retrieve the password from the file named argument.

Note:

The password shouldn't be specified on the command line or in a script unless it is for testing purposes, or you are on a secure system.

-storetype storetype

Specifies the type of keystore to be instantiated. The default keystore type is the one that is specified as the value of the keystore.type property in the security properties file, which is returned by the static getDefaultType method in java.security.KeyStore.

The PIN for a PKCS #11 token can also be specified with the -storepass option. If none is specified, then the keytool and jarsigner commands prompt for the token PIN. If the token has a protected authentication path (such as a dedicated PIN-pad or a biometric reader), then the -protected option must be specified and no password options can be specified.

-keypass [:env | :file] argument -certchain file

Specifies the password used to protect the private key of the keystore entry addressed by the alias specified on the command line. The password is required when using jarsigner to sign a JAR file. If no password is provided on the command line, and the required password is different from the store password, then the user is prompted for it.

If the modifier env or file isn't specified, then the password has the value argument. Otherwise, the password is retrieved as follows:

env: Retrieve the password from the environment variable named argument.

file: Retrieve the password from the file named argument.

Note:

The password shouldn't be specified on the command line or in a script unless it is for testing purposes, or you are on a secure system.

-certchain file

Specifies the certificate chain to be used when the certificate chain associated with the private key of the keystore entry that is addressed by the alias specified on the command line isn't complete. This can happen when the keystore is located on a hardware token where there isn't enough capacity to hold a complete certificate chain. The file can be a sequence of concatenated X.509 certificates, or a single PKCS#7 formatted data block, either in binary encoding format or in printable encoding format (also known as Base64 encoding) as defined by Internet RFC 1421 Certificate Encoding Standard.

-sigfile file

Specifies the base file name to be used for the generated .SF and .DSA files. For example, if file is DUKESIGN, then the generated .SF and .DSA files are named DUKESIGN.SF and DUKESIGN.DSA, and placed in the META-INF directory of the signed JAR file.

The characters in the file must come from the set a-zA-Z0-9\_-. Only letters, numbers, underscore, and hyphen characters are allowed. All lowercase characters are converted to uppercase for the .SF and .DSA file names.

If no -sigfile option appears on the command line, then the base file name for the .SF and .DSA files is the first 8 characters of the alias name specified on the command line, all converted to upper case. If the alias name has fewer than 8 characters, then the full alias name is used. If the alias name contains any characters that aren't valid in a signature file name, then each such character is converted to an underscore (\_) character to form the file name.

-signedjar file

Specifies the name of signed JAR file.

-digestalg algorithm

Specifies the name of the message digest algorithm to use when digesting the entries of a JAR file.

For a list of standard message digest algorithm names, see Java Security Standard Algorithm Names.

If this option isn't specified, then SHA256 is used. There must either be a statically installed provider supplying an implementation of the specified algorithm or the user must specify one with the -addprovider or -providerClass options; otherwise, the command will not succeed.

-sigalg algorithm

Specifies the name of the signature algorithm to use to sign the JAR file.

This algorithm must be compatible with the private key used to sign the JAR file. If this option isn't specified, then use a default algorithm matching the private key as described in the Supported Algorithms section. There must either be a statically installed provider supplying an implementation of the specified algorithm or you must specify one with the -addprovider or -providerClass option; otherwise, the command doesn't succeed.

For a list of standard message digest algorithm names, see Java Security Standard Algorithm Names.

-verify

Verifies a signed JAR file.

-verbose[:suboptions]

When the -verbose option appears on the command line, it indicates that the jarsigner use the verbose mode when signing or verifying with the suboptions determining how much information is shown. This causes the , which causes jarsigner to output extra information about the progress of the JAR signing or verification. The suboptions can be all, grouped, or summary.

If the -certs option is also specified, then the default mode (or suboption all) displays each entry as it is being processed, and after that, the certificate information for each signer of the JAR file.

If the -certs and the -verbose:grouped suboptions are specified, then entries with the same signer info are grouped and displayed together with their certificate information.

If -certs and the -verbose:summary suboptions are specified, then entries with the same signer information are grouped and displayed together with their certificate information.

Details about each entry are summarized and displayed as one entry (and more). See Example of Verifying a Signed JAR File and Example of Verification with Certificate Information.

-certs

If the -certs option appears on the command line with the -verify and -verbose options, then the output includes certificate information for each signer of the JAR file. This information includes the name of the type of certificate (stored in the .DSA file) that certifies the signer's public key, and if the certificate is an X.509 certificate (an instance of the java.security.cert.X509Certificate), then the distinguished name of the signer.

The keystore is also examined. If no keystore value is specified on the command line, then the default

keystore file (if any) is checked. If the public key certificate for a signer matches an entry in the keystore, then the alias name for the keystore entry for that signer is displayed in parentheses.

-revCheck

This option enables revocation checking of certificates when signing or verifying a JAR file. The jarsigner command attempts to make network connections to fetch OCSP responses and CRLs if the -revCheck option is specified on the command line. Note that revocation checks are not enabled unless this option is specified.

-tsa url

If -tsa http://example.tsa.url appears on the command line when signing a JAR file then a time stamp is generated for the signature. The URL, http://example.tsa.url, identifies the location of the Time Stamping Authority (TSA) and overrides any URL found with the -tsacert option. The -tsa option doesn't require the TSA public key certificate to be present in the keystore.

To generate the time stamp, jarsigner communicates with the TSA with the Time-Stamp Protocol (TSP) defined in RFC 3161. When successful, the time stamp token returned by the TSA is stored with the signature in the signature block file.

-tsacert alias

When -tsacert alias appears on the command line when signing a JAR file, a time stamp is generated for the signature. The alias identifies the TSA public key certificate in the keystore that is in effect. The entry's certificate is examined for a Subject Information Access extension that contains a URL identifying the location of the TSA.

The TSA public key certificate must be present in the keystore when using the -tsacert option.

-tsapolicyid policyid

Specifies the object identifier (OID) that identifies the policy ID to be sent to the TSA server. If this option isn't specified, no policy ID is sent and the TSA server will choose a default policy ID.

Object identifiers are defined by X.696, which is an ITU Telecommunication Standardization Sector (ITU-T) standard. These identifiers are typically period-separated sets of non-negative digits like 1.2.3.4, for example.

-tsadigestalg algorithm

Specifies the message digest algorithm that is used to generate the message imprint to be sent to the TSA server. If this option isn't specified, SHA-256 will be used.

See Supported Algorithms.

For a list of standard message digest algorithm names, see Java Security Standard Algorithm Names.

-internalsf

In the past, the .DSA (signature block) file generated when a JAR file was signed included a complete encoded copy of the .SF file (signature file) also generated. This behavior has been changed. To reduce the overall size of the output JAR file, the .DSA file by default doesn't contain a copy of the .SF file anymore. If -internalsf appears on the command line, then the old behavior is utilized. This option is useful for testing. In practice, don't use the -internalsf option because it incurs higher overhead.

-sectionsonly

If the -sectionsonly option appears on the command line, then the .SF file (signature file) generated when a JAR file is signed doesn't include a header that contains a hash of the whole manifest file. It contains only the information and hashes related to each individual source file included in the JAR file. See Signature File.

By default, this header is added, as an optimization. When the header is present, whenever the JAR file is verified, the verification can first check to see whether the hash in the header matches the hash of the whole manifest file. When there is a match, verification proceeds to the next step. When there is no match, it is necessary to do a less optimized verification that the hash in each source file information section in the .SF file equals the hash of its corresponding section in the manifest file. See JAR File Verification.

The -sectionsonly option is primarily used for testing. It shouldn't be used other than for testing because using it incurs higher overhead.

-protected

Values can be either true or false. Specify true when a password must be specified through a protected authentication path such as a dedicated PIN reader.

-providerName providerName

If more than one provider was configured in the java.security security properties file, then you can use the -providerName option to target a specific provider instance. The argument to this option is the name of the provider.

For the Oracle PKCS #11 provider, providerName is of the form SunPKCS11-TokenName, where TokenName is the name suffix that the provider instance has been configured with, as detailed in the configuration attributes table. For example, the following command lists the contents of the PKCS #11 keystore provider instance with name suffix SmartCard:

jarsigner -keystore NONE -storetype PKCS11 -providerName SunPKCS11-SmartCard -list

-addprovider name [-providerArg arg]

Adds a security provider by name (such as SunPKCS11) and an optional configure argument. The value of the security provider is the name of a security provider that is defined in a module.

Used with the -providerArg ConfigFilePath option, the keytool and jarsigner tools install the provider dynamically and use ConfigFilePath for the path to the token configuration file. The following example shows a command to list a PKCS #11 keystore when the Oracle PKCS #11 provider wasn't configured in the security properties file.

jarsigner -keystore NONE -storetype PKCS11 -addprovider SunPKCS11 -providerArg /mydir1/mydir2/token.config

-providerClass provider-class-name [-providerArg arg]

Used to specify the name of cryptographic service provider's master class file when the service provider isn't listed in the java.security security properties file. Adds a security provider by fully-qualified class name and an optional configure argument.

Note:

The preferred way to load PKCS11 is by using modules. See -addprovider.

-Jjavaoption

Passes through the specified javaoption string directly to the Java interpreter. The jarsigner command is a wrapper around the interpreter. This option shouldn't contain any spaces. It is useful for adjusting the execution environment or memory usage. For a list of possible interpreter options, type java -h or java -X at the command line.

-strict

During the signing or verifying process, the command may issue warning messages. If you specify this option, the exit code of the tool reflects the severe warning messages that this command found. See Errors and Warnings.

-conf url

Specifies a pre-configured options file. Read the keytool documentation for details. The property keys supported are "jarsigner.all" for all actions, "jarsigner.sign" for signing, and "jarsigner.verify" for verification. jarsigner arguments including the JAR file name and alias name(s) cannot be set in this file.

### Example

## javadoc

### Description

* + - 1. The javadoc tool parses the declarations and documentation comments in a set of Java source files and produces corresponding HTML pages that describe (by default) the public and protected classes, nested classes (but not anonymous inner classes), interfaces, constructors, methods, and fields.
      2. You can use the javadoc tool to generate the API documentation or the implementation documentation for a set of source files.
      3. You can run the javadoc tool on entire packages, individual source files, or both.
      4. When documenting entire packages, you can use the -subpackages option either to recursively traverse a directory and its subdirectories, or to pass in an explicit list of package names.
      5. When you document individual source files, pass in a list of Java source file names.
      6. See javadoc Overview in Java Platform, Standard Edition Javadoc Guide for information about using the javadoc tool.

### Synopsis

javadoc [options] [packagenames] [sourcefiles] [@files]

options

Specifies command-line options, separated by spaces. See Options for javadoc, Extended Options, Standard doclet Options, and Additional Options Provided by the Standard doclet.

packagenames

Specifies names of packages that you want to document, separated by spaces,

for example java.lang java.lang.reflect java.awt.

If you want to also document the subpackages, then use the -subpackages option to specify the

packages.

By default, javadoc looks for the specified packages in the current directory and subdirectories.

Use the -sourcepath option to specify the list of directories where to look for packages.

sourcefiles

Specifies names of Java source files that you want to document, separated by spaces,

for example Class.java Object.java Button.java.

By default, javadoc looks for the specified classes in the current directory.

However, you can specify the full path to the class file and use wildcard characters,

for example /home/src/java/awt/Graphics\*.java.

You can also specify the path relative to the current directory.

@files

Specifies names of files that contain a list of javadoc tool options, package names, and source file names in any order.

### Options for javadoc

* + - 1. The following core javadoc options are equivalent to corresponding javac options.

See Standard Options in javac for the detailed descriptions of using these options:

--add-modules

-bootclasspath

--class-path, -classpath, or -cp

--enable-preview

-encoding

-extdirs

--limit-modules

--module

--module-path or -p

--module-source-path

--release

--source or -source

--source-path or -sourcepath

--system

--upgrade-module-path

* + - 1. The following options are the core javadoc options that are not equivalent to a corresponding javac option:

|  |  |
| --- | --- |
| -breakiterator | 1. Computes the first sentence with BreakIterator. 2. The first sentence is copied to the package, class, or member summary and to the alphabetic index. 3. The BreakIterator class is used to determine the end of a sentence for all languages except for English. 4. English default sentence-break algorithm --- Stops at a period followed by a space or an HTML block tag, such as <P>. 5. Breakiterator sentence-break algorithm --- Stops at a period, question mark, or exclamation point followed by a space when the next word starts with a capital letter.    1. This is meant to handle most abbreviations (such as "The serial no. is valid", but will not handle "Mr. Smith").    2. The -breakiterator option doesn't stop at HTML tags or sentences that begin with numbers or symbols.    3. The algorithm stops at the last period in ../filename, even when embedded in an HTML tag. |
| -doclet class | Generates output by using an alternate doclet.  Use the fully qualified name.  This doclet defines the content and formats the output.  If the -doclet option isn't used, then the javadoc tool uses the standard doclet for generating the default HTML format.  This class must contain the start(Root) method.  The path to this starting class is defined by the -docletpath option. |
| -docletpath path | Specifies where to find doclet class files (specified with the -doclet option) and any JAR files it depends on.  If the starting class file is in a JAR file, then this option specifies the path to that JAR file.  You can specify an absolute path or a path relative to the current directory.  If classpathlist contains multiple paths or JAR files, then they should be separated with a colon (:) on Linux.  This option isn't necessary when the doclet starting class is already in the search path. |
| -exclude pkglist | Unconditionally, excludes the specified packages and their subpackages from the list formed by -subpackages.  It excludes those packages even when they would otherwise be included by some earlier or later -subpackages option.  The following example would include java.io, java.util, and java.math (among others), but would exclude packages rooted at java.net and java.lang.  Notice that these examples exclude java.lang.ref, which is a subpackage of java.lang.  javadoc -sourcepath /home/user/src -subpackages java -exclude java.net:java.lang |
| --expand-requires value | Instructs the javadoc tool to expand the set of modules to be documented.  By default, only the modules given explicitly on the command line are documented.  Supports the following values:  transitive: additionally includes all the required transitive dependencies of thosemodules.  all: includes all dependencies. |
| --help  -help  -h  -? | Prints a synopsis of the standard options. |
| --help-extra  -X | Prints a synopsis of the set of extra options. |
| -Jflag | Passes flag directly to the Java Runtime Environment (JRE) that runs the javadoc tool.  For example, if you must ensure that the system sets aside 32 MB of memory in which to process the generated documentation, then you would call the -Xmx option as follows:  javadoc -J-Xmx32m -J-Xms32m com.mypackage.  Be aware that -Xms is optional because it only sets the size of initial memory, which is useful when you know the minimum amount of memory required.  There is no space between the J and the flag.  Use the -version option to report the version of the JRE being used to run the javadoc tool.  javadoc -J-version  java version "10-ea" 2018-03-20  Java(TM) SE Runtime Environment 18.3 (build 10-ea+36)  Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10-ea+36, mixed mode) |
| -locale name | Specifies the locale that the javadoc tool uses when it generates documentation.  The argument is the name of the locale, as described in java.util.Locale documentation, such as en\_US (English, United States) or en\_US\_WIN (Windows variant).  Note:  The -locale option must be placed ahead (to the left) of any options provided by the standard doclet or any other doclet.  Otherwise, the navigation bars appear in English. This is the only command-line option that depends on order.  Specifying a locale causes the javadoc tool to choose the resource files of that locale for messages such as strings in the navigation bar, headings for lists and tables, help file contents, comments in the stylesheet.css file, and so on.  It also specifies the sorting order for lists sorted alphabetically, and the sentence separator to determine the end of the first sentence.  The -locale option doesn't determine the locale of the documentation comment text specified in the source files of the documented classes. |
| -package | Shows only package, protected, and public classes and members. |
| -private | Shows all classes and members. |
| -protected | Shows only protected and public classes and members. This is the default. |
| -public | Shows only the public classes and members. |
| -quiet | Shuts off messages so that only the warnings and errors appear to make them easier to view. It also suppresses the version string. |
| --show-members value | Specifies which members (fields or methods) are documented, where value can be any of the following:  protected: The default value is protected.  public: Shows only public values.  package: Shows public, protected, and package members.  private: Shows all members. |
| --show-module-contents value | Specifies the documentation granularity of module declarations, where value can be api or all. |
| --show-packages value | Specifies which modules packages are documented, where value can be exported or all packages. |
| --show-types value | Specifies which types (classes, interfaces, etc.) are documented, where value can be any of the following:  protected: The default value. Shows public and protected types.  public: Shows only public values.  package: Shows public, protected, and package types.  private: Shows all types. |
| -subpackages subpkglist | Generates documentation from source files in the specified packages and recursively in their subpackages.  This option is useful when adding new subpackages to the source code because they are automatically included.  Each package argument is any top-level subpackage (such as java) or fully qualified package (such as javax.swing) that doesn't need to contain source files.  Arguments are separated by colons on all operating systems.  Wild cards aren't allowed.  Use -sourcepath to specify where to find the packages.  This option doesn't process source files that are in the source tree but don't belong to the packages.  For example,  the following commands generates documentation for packages named java and javax.swing and all of their subpackages.  javadoc -d docs -sourcepath /home/user/src -subpackages java:javax.swing |
| -verbose | Provides more detailed messages while the javadoc tool runs.  Without the -verbose option, messages appear for loading the source files, generating the documentation (one message per source file), and sorting.  The -verbose option causes the printing of additional messages that specify the number of milliseconds to parse each Java source file. |
| --version | Prints version information. |
| -Werror | Reports an error if any warnings occur. |

### Extended Options

* + - 1. The extended options for javadoc are subject to change without notice.
      2. The following extended javadoc options are equivalent to corresponding javac options.
      3. See Extra Options in javac for the detailed descriptions of using these options:

--add-exports

--add-reads

--patch-module

-Xmaxerrs

-Xmaxwarns

### Standard doclet Options

|  |  |
| --- | --- |
| --add-stylesheet file | Adds additional stylesheet file for the generated documentation.  This option can be used one or more times to specify additional stylesheets included in the documentation.  Command-line example:  javadoc --add-stylesheet new\_stylesheet\_1.css --add-stylesheet new\_stylesheet\_2.css pkg\_foo |
| --allow-script-in-comments | Allow JavaScript in options and comments |
| -author | Includes the @author text in the generated docs. |
| -bottom html-code | Specifies the text to be placed at the bottom of each output file.  The text is placed at the bottom of the page, underneath the lower navigation bar.  The text can contain HTML tags and white space, but when it does, the text must be enclosed in quotation marks.  Use escape characters for any internal quotation marks within text. |
| -charset name | Specifies the HTML character set for this document. The name should be a preferred MIME name as specified in the IANA Registry, Character Sets.  For example:  javadoc -charset "iso-8859-1" mypackage  This command inserts the following line in the head of every generated page:  <META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  The META tag is described in the HTML standard (4197265 and 4137321), HTML Document Representation. |
| -d directory | Specifies the destination directory where the javadoc tool saves the generated HTML files.  If you omit the -d option, then the files are saved to the current directory.  The directory value can be absolute or relative to the current working directory.  The destination directory is automatically created when the javadoc tool runs.  For example, the following command generates the documentation for the package com.mypackage and saves the results in the /user/doc/ directory:  javadoc -d /user/doc/ com.mypackage |
| -docencoding name | Specifies the encoding of the generated HTML files.  The name should be a preferred MIME name as specified in the IANA Registry, Character Sets.  Three options are available for use in a javadoc encoding command.  The -encoding option is used for encoding the files read by the javadoc tool, while the -docencoding and -charset options are used for encoding the files written by the tool.  Of the three available options, at most, only the input and an output encoding option are used in a single encoding command.  If you specify both input and output encoding options in a command, they must be the same value.  If you specify neither output option, it the tool defaults to the input encoding.  For example:  javadoc -docencoding "iso-8859-1" mypackage |
| -docfilessubdirs | Recursively copies doc-file subdirectories. |
| -doctitle html-code | Specifies the title to place near the top of the overview summary file.  The text specified in the title tag is placed as a centered, level-one heading directly beneath the top navigation bar.  The title tag can contain HTML tags and white space, but when it does, you must enclose the title in quotation marks.  Additional quotation marks within the title tag must be escaped.  For example, javadoc -header "<b>My Library</b><br>v1.0" com.mypackage. |
| -excludedocfilessubdir name | Excludes any doc files sub directories with the given name.  Enables deep copying of doc-files directories.  Subdirectories and all contents are recursively copied to the destination.  For example, the directory doc-files/example/images and all of its contents are copied.  There is also an option to exclude subdirectories. |
| -footer html-code | Specifies the footer text to be placed at the bottom of each output file.  Thehtml-code value is placed to the right of the lower navigation bar.  The html-code value can contain HTML tags and white space, but when it does, the html-code value must be enclosed in quotation marks.  Use escape characters for any internal quotation marks within a footer. |
| -group namep1:p2 | Group the specified packages together in the Overview page. |
| -header html-code | Specifies the header text to be placed at the top of each output file.  The header is placed to the right of the upper navigation bar.  The header can contain HTML tags and white space, but when it does, the header must be enclosed in quotation marks.  Use escape characters for internal quotation marks within a header.  For example, javadoc -header "<b>My Library</b><br>v1.0" com.mypackage. |
| -helpfile filename | Includes the file that links to the HELP link in the top and bottom navigation bars .  Without this option, the javadoc tool creates a help file help-doc.html that is hard-coded in the javadoc tool.  This option lets you override the default.  The filename can be any name and isn't restricted to help-doc.html.  The javadoc tool adjusts the links in the navigation bar accordingly.  For example:  javadoc -helpfile /home/user/myhelp.html java.awt. |
| -html5 | This option is a no-op and is just retained for backwards compatibility. |
| --javafx or -javafx | Enables JavaFX functionality. |
| -keywords | Adds HTML keyword <META> tags to the generated file for each class.  These tags can help search engines that look for <META> tags find the pages.  Most search engines that search the entire Internet don't look at <META> tags, because pages can misuse them.  Search engines offered by companies that confine their searches to their own website can benefit by looking at <META> tags.  The <META> tags include the fully qualified name of the class and the unqualified names of the fields and methods.  Constructors aren't included because they are identical to the class name.  For example, the class String starts with these keywords:  <META NAME="keywords" CONTENT="java.lang.String class">  <META NAME="keywords" CONTENT="CASE\_INSENSITIVE\_ORDER">  <META NAME="keywords" CONTENT="length()">  <META NAME="keywords" CONTENT="charAt()"> |
| -link url | Creates links to existing javadoc generated documentation of externally referenced classes.  The url argument is the absolute or relative URL of the directory that contains the external javadoc generated documentation.  You can specify multiple -link options in a specified javadoc tool run to link to multiple documents.  Either a package-list or an element-list file must be in this url directory (otherwise, use the -linkoffline option).  Note:  The package-list and element-list files are generated by the javadoc tool when generating the API documentation and should not be modified by the user.  When you use the javadoc tool to document packages, it uses the package-list file to determine the packages declared in an API.  When you generate API documents for modules, the javadoc tool uses the element-list file to determine the modules and packages declared in an API.  The javadoc tool reads the names from the appropriate list file and then links to the packages or modules at that URL.  When the javadoc tool runs, the url value is copied into the <A HREF> links that are created. Therefore, url must be the URL to the directory and not to a file.  You can use an absolute link for url to enable your documents to link to a document on any web site, or you can use a relative link to link only to a relative location.  If you use a relative link, then the value you pass in should be the relative path from the destination directory (specified with the -d option) to the directory containing the packages being linked to.  When you specify an absolute link, you usually use an HTTP link.  However, if you want to link to a file system that has no web server, then you can use a file link. Use a file link only when everyone who wants to access the generated documentation shares the same file system.  In all cases, and on all operating systems, use a slash as the separator, whether the URL is absolute or relative, and https:, http:, or file: as specified in the URL Memo: Uniform Resource Locators.  -link https://<host>/<directory>/<directory>/.../<name>  -link http://<host>/<directory>/<directory>/.../<name>  -link file://<host>/<directory>/<directory>/.../<name>  -link <directory>/<directory>/.../<name> |
| -linkoffline url1 url2 | This option is a variation of the -link option. They both create links to javadoc generated documentation for externally referenced classes. You can specify multiple -linkoffline options in a specified javadoc tool run.  Use the -linkoffline option when:  Linking to a document on the web that the javadoc tool can't access through a web connection  The package-list or element-list file of the external document either isn't accessible or doesn't exist at the URL location, but does exist at a different location and can be specified by either the package-list or element-list file (typically local).  Note:  The package-list and element-list files are generated by the javadoc tool when generating the API documentation and should not be modified by the user.  If url1 is accessible only on the World Wide Web, then the -linkoffline option removes the constraint that the javadoc tool must have a web connection to generate documentation.  Another use of the -linkoffline option is as a work-around to update documents. After you have run the javadoc tool on a full set of packages or modules, you can run the javadoc tool again on a smaller set of changed packages or modules, so that the updated files can be inserted back into the original set.  For example  the -linkoffline option takes two arguments.  The first is for the string to be embedded in the <a href> links, and the second tells the javadoc tool where to find either the package-list or element-list file.  The url1 or url2 value is the absolute or relative URL of the directory that contains the external javadoc generated documentation that you want to link to.  When relative, the value should be the relative path from the destination directory (specified with the -d option) to the root of the packages being linked to.  See url in the -link option. |
| -linksource | Creates an HTML version of each source file (with line numbers) and adds links to them from the standard HTML documentation.  Links are created for classes, interfaces, constructors, methods, and fields whose declarations are in a source file.  Otherwise, links aren't created, such as for default constructors and generated classes.  This option exposes all private implementation details in the included source files, including private classes, private fields, and the bodies of private methods, regardless of the -public, -package, -protected, and -private options.  Unless you also use the -private option, not all private classes or interfaces are accessible through links.  Each link appears on the name of the identifier in its declaration.  For example, the link to the source code of the Button class would be on the word Button:  public class Button extends Component implements Accessible  The link to the source code of the getLabel method in the Button class is on the word getLabel:  public String getLabel() |
| --main-stylesheet file  -stylesheetfile file | Specifies the path of an alternate stylesheet file that contains the definitions for the CSS styles used in the generated documentation.  This option lets you override the default.  If you do not specify the option, the javadoc tool will create and use a default stylesheet.  The file name can be any name and isn't restricted to stylesheet.css.  The --main-stylesheet option is the preferred form.  Command-line example:  javadoc --main-stylesheet main\_stylesheet.css pkg\_foo |
| -nocomment | Suppresses the entire comment body, including the main description and all tags, and generate only declarations.  This option lets you reuse source files that were originally intended for a different purpose so that you can produce skeleton HTML documentation during the early stages of a new project. |
| -nodeprecated | Prevents the generation of any deprecated API in the documentation.  This does what the -nodeprecatedlist option does, and it doesn't generate any deprecated API throughout the rest of the documentation.  This is useful when writing code when you don't want to be distracted by the deprecated code. |
| -nodeprecatedlist | Prevents the generation of the file that contains the list of deprecated APIs (deprecated-list.html) and the link in the navigation bar to that page.  The javadoc tool continues to generate the deprecated API throughout the rest of the document.  This is useful when your source code contains no deprecated APIs, and you want to make the navigation bar cleaner. |
| --no-frames | This option is a no-op and is just retained for backwards compatibility. |
| -nohelp | Omits the HELP link in the navigation bars at the top and bottom of each page of output. |
| -noindex | Omits the index from the generated documents. The index is produced by default. |
| -nonavbar | Prevents the generation of the navigation bar, header, and footer, that are usually found at the top and bottom of the generated pages.  The -nonavbar option has no affect on the -bottom option.  The -nonavbar option is useful when you are interested only in the content and have no need for navigation, such as when you are converting the files to PostScript or PDF for printing only. |
| -noqualifier name1:name2... | Excludes the list of qualifiers from the output.  The package name is removed from places where class or interface names appear.  The following example omits all package qualifiers:  -noqualifier all.  The following example omits java.lang and java.io package qualifiers:  -noqualifier java.lang:java.io.  The following example omits package qualifiers starting with java and com.sun subpackages, but not javax:  -noqualifier java.\*:com.sun.\*.  Where a package qualifier would appear due to the previous behavior, the name can be suitably shortened.  This rule is in effect whether or not the -noqualifier option is used. |
| -nosince | Omits from the generated documents the Since sections associated with the @since tags. |
| -notimestamp | Suppresses the time stamp, which is hidden in an HTML comment in the generated HTML near the top of each page.  The -notimestamp option is useful when you want to run the javadoc tool on two source bases and get the differences between diff them, because it prevents time stamps from causing a diff (which would otherwise be a diff on every page).  The time stamp includes the javadoc tool release number. |
| -notree | Omits the class and interface hierarchy pages from the generated documents.  These are the pages you reach using the Tree button in the navigation bar.  The hierarchy is produced by default. |
| --override-methods (detail|summary) | Documents overridden methods in the detail or summary sections. |
| -overview filename | Specifies that the javadoc tool should retrieve the text for the overview documentation from the source file specified by filename and place it on the Overview page (overview-summary.html).  A relative path specified with the file name is relative to the current working directory.  While you can use any name you want for the filename value and place it anywhere you want for the path, it is typical to name it overview.html and place it in the source tree at the directory that contains the topmost package directories.  In this location, no path is needed when documenting packages, because the -sourcepath option points to this file.  Linux and OS X: For example, if the source tree for the java.lang package is /src/classes/java/lang/, then you could place the overview file at /src/classes/overview.html.  The overview page is created only when you pass two or more package names to the javadoc tool.  The title on the overview page is set by -doctitle. |
| -serialwarn | Generates compile-time warnings for missing @serial tags.  By default, Javadoc generates no serial warnings.  Use this option to display the serial warnings, which helps to properly document default serializable fields and writeExternal methods. |
| -sourcetab tablength | Specifies the number of spaces each tab uses in the source. |
| -splitindex | Splits the index file into multiple files, alphabetically, one file per letter, plus a file for any index entries that start with non-alphabetical symbols. |
| -tag name:locations:header | Specifies single argument custom tags.  For the javadoc tool to spell-check tag names, it is important to include a -tag option for every custom tag that is present in the source code, disabling (with X) those that aren't being output in the current run.  The colon (:) is always the separator.  The -tag option outputs the tag heading, header, in bold, followed on the next line by the text from its single argument.  Similar to any block tag, the argument text can contain inline tags, which are also interpreted.  The output is similar to standard one-argument tags, such as the @return and @author tags.  Omitting a header value causes the name to be the heading. |
| -taglet class | Specifies the fully qualified name of the taglet used in generating the documentation for that tag.  Use the fully qualified name for the class value.  This taglet also defines the number of text arguments that the custom tag has.  The taglet accepts those arguments, processes them, and generates the output.  Taglets are useful for block or inline tags.  They can have any number of arguments and implement custom behavior, such as making text bold, formatting bullets, writing out the text to a file, or starting other processes.  Taglets can only determine where a tag should appear and in what form.  All other decisions are made by the doclet.  A taglet can't do things such as remove a class name from the list of included classes.  However, it can execute side effects, such as printing the tag's text to a file or triggering another process.  Use the -tagletpath option to specify the path to the taglet.  The following example inserts the To Do taglet after Parameters and ahead of Throws in the generated pages.  -taglet com.sun.tools.doclets.ToDoTaglet  -tagletpath /home/taglets  -tag return  -tag param  -tag todo  -tag throws  -tag see  Alternately, you can use the -taglet option in place of its -tag option, but that might be difficult to read. |
| -tagletpath tagletpathlist | Specifies the search paths for finding taglet class files.  The tagletpathlist can contain multiple paths by separating them with a colon (:).  The javadoc tool searches all subdirectories of the specified paths. |
| -top html-code | Specifies the text to be placed at the top of each output file. |
| -use | Creates class and package usage pages.  Includes one Use page for each documented class and package.  The page describes what packages, classes, methods, constructors and fields use any API of the specified class or package.  Given class C, things that use class C would include subclasses of C, fields declared as C, methods that return C, and methods and constructors with parameters of type C.  For example, you can look at the Use page for the String type.  Because the getName method in the java.awt.Font class returns type String, the getName method uses String and so the getName method appears on the Use page for String.  This documents only uses of the API, not the implementation.  When a method uses String in its implementation, but doesn't take a string as an argument or return a string, that isn't considered a use of String.To access the generated Use page, go to the class or package and click the Use link in the navigation bar. |
| -version | Includes the version text in the generated docs.  This text is omitted by default.  To find out what version of the javadoc tool you are using, use the -J-version option. |
| -windowtitle title | Specifies the title to be placed in the HTML <title> tag.  The text specified in the title tag appears in the window title and in any browser bookmarks (favorite places) that someone creates for this page.  This title shouldn't contain any HTML tags because the browser doesn't interpret them correctly.  Use escape characters on any internal quotation marks within the title tag.  If the -windowtitle option is omitted, then the javadoc tool uses the value of the -doctitle option for the -windowtitle option.  For example, javadoc -windowtitle "My Library" com.mypackage. |

### Additional Options Provided by the Standard Doclet

|  |  |
| --- | --- |
| -Xdoclint | Enables recommended checks for problems in documentation comments. |
| -Xdoclint:(all|none|[-]group) | 1 Enable or disable specific checks for bad references, accessibility issues, missing documentation comments, errors in documentation comment syntax and missing HTML tags.  2 This option enables the javadoc tool to check for all documentation comments included in the generated output.  3 You can select which items to include in the generated output with the standard options -public, -protected, -package and -private.  4 When the -Xdoclint option is enabled, it reports issues with messages similar to the javac command.  5 The javadoc tool prints a message, a copy of the source line, and a caret pointing at the exact position where the error was detected.  6 Messages may be either warnings or errors, depending on their severity and the likelihood to cause an error if the generated documentation were to be run through a validator.  For example:  missing documentation comments, duplicate information, and extraneous comments do not cause the javadoc tool to generate invalid HTML, so these issues are reported as warnings; syntax errors, missing required HTML end tags, and references to missing or misspelled elements cause the javadoc tool to generate invalid output, so these issues are reported as errors.  7 -Xdoclint option validates input comments based upon the requested markup.  8 By default, the -Xdoclint option is enabled. Disable it with the option -Xdoclint:none.  9 The following options change what the -Xdoclint option reports:  -Xdoclint none: Disables the -Xdoclint option  -Xdoclint group: Enables group checks  -Xdoclint all: Enables all groups of checks  -Xdoclint all,-group: Enables all checks except group checks  10 The group variable has one of the following values:  10.1 accessibility: Checks for the issues to be detected by an accessibility checker (for example, no caption or summary attributes specified in a <table> tag).  10.2 html: Detects high-level HTML issues, such as putting block elements inside inline elements, or not closing elements that require an end tag. The rules are derived from the HTML 4 Specification or the HTML 5 Specification based on the standard doclet html output generation selected. This type of check enables the javadoc tool to detect HTML issues that some browsers might not interpret as intended.  10.3 missing: Checks for missing documentation comments or tags (for example, a missing comment or class, or a missing @return tag or similar tag on a method).  10.4 reference: Checks for issues relating to the references to Java API elements from documentation comment tags (for example, item not found in @see, or a bad name after @param).  10.5 syntax: Checks for low level issues like unescaped angle brackets (< and >) and ampersands (&) and invalid documentation comment tags.  11 You can specify the -Xdoclint option multiple times to enable the option to check errors and warnings in multiple categories. Alternatively, you can specify multiple error and warning categories by using the preceding options. For example, use either of the following commands to check for the HTML, syntax, and accessibility issues in the file filename.  javadoc -Xdoclint:html -Xdoclint:syntax -Xdoclint:accessibility filename  javadoc -Xdoclint:html,syntax,accessibility filename  12 Note:  12.1 The javadoc tool doesn't guarantee the completeness of these checks. In particular, it isn't a full HTML compliance checker.  12.2 The goal of the -Xdoclint option is to enable the javadoc tool to report majority of common errors.  12.3 The javadoc tool doesn't attempt to fix invalid input, it just reports it. |
| -Xdoclint/package:[-]packages | Enables or disables checks in specific packages.  packages is a comma separated list of package specifiers.  A package specifier is either a qualified name of a package or a package name prefix followed by \*, which expands to all sub packages of the given package.  Prefix the package specifier with - to disable checks for the specified packages. |
| -Xdocrootparent url | Replaces all @docRoot items followed by/.. in Javadoc comments with the url. |

### Conformance

* + - 1. The standard doclet does not validate the content of documentation comments for conformance, nor does it attempt to correct any errors in documentation comments.
      2. Anyone running javadoc is advised to be aware of the problems that may arise when generating non-conformant output or output containing executable content, such as JavaScript.
      3. The standard doclet does provide the doclint feature to help developers detect common problems in documentation comments; but it is also recommended to check the generated output with any appropriate conformance and other checking tools.
      4. For more details on the conformance requirements for HTML5 documents, see Conformance requirements in the HTML5 Specification.
      5. For more details on security issues related to web pages, see the Open Web Application Security Project (OWASP) page.

### Example

## jconsole

### Description

The jconsole command starts a graphical console tool that lets you monitor and manage Java applications and virtual machines on a local or remote machine.

### Synopsis

jconsole [-interval=n] [-notile] [-plugin path] [-version] [connection ... ] [-Jinput\_arguments]

jconsole -help

-interval

Sets the update interval to n seconds (default is 4 seconds).

-notile

Doesn't tile the windows for two or more connections.

-pluginpath path

Specifies the path that jconsole uses to look up plug-ins.

The plug-in path should contain a provider-configuration file named META-INF/services/com.sun.tools.jconsole.JConsolePlugin that contains one line for each plug-in.

The line specifies the fully qualified class name of the class implementing the com.sun.tools.jconsole.JConsolePlugin class.

-version

Prints the program version.

connection = pid | host:port | jmxURL

A connection is described by either pid, host:port or jmxURL.

The pid value is the process ID of a target process.

The JVM must be running with the same user ID as the user ID running the jconsole command.

The host:port values are the name of the host system on which the JVM is running, and the port number specified by the system property com.sun.management.jmxremote.port when the JVM was started.

The jmxUrl value is the address of the JMX agent to be connected to as described in JMXServiceURL.

-Jinput\_arguments

Passes input\_arguments to the JVM on which the jconsole command is run.

-help or --help

Displays the help message for the command.

## jdb

### Description

* + - 1. The Java Debugger (JDB) is a simple command-line debugger for Java classes.
      2. The jdb command and its options call the JDB.
      3. The jdb command demonstrates the Java Platform Debugger Architecture and provides inspection and debugging of a local or remote JVM.

### Synopsis

jdb [options] [classname] [arguments]

options

This represents the jdb command-line options. See Options for the jdb command.

classname

This represents the name of the main class to debug.

arguments

This represents the arguments that are passed to the main() method of the class.

### Options for the jdb command

* + - 1. The following options are accepted by the jdb command:

|  |  |
| --- | --- |
| -sourcepath dir1:dir2:... | Uses the specified path to search for source files in the specified path.  If this option is not specified, then use the default path of dot (.). |
| -attach address | Attaches the debugger to a running JVM with the default connection mechanism. |
| -listen address | Waits for a running JVM to connect to the specified address with a standard connector. |
| -listenany | Waits for a running JVM to connect at any available address using a standard connector. |
| -launch | Starts the debugged application immediately upon startup of the jdb command.  The -launch option removes the need for the run command.  The debugged application is launched and then stopped just before the initial application class is loaded.  At that point, you can set any necessary breakpoints and use the cont command to continue execution. |
| -listconnectors | Lists the connectors available in this JVM. |
| -connect connector-name:name1=value1.... | Connects to the target JVM with the named connector and listed argument values. |
| -dbgtrace [flags] | Prints information for debugging the jdb command. |
| -tclient | Runs the application in the Java HotSpot VM client. |
| -tserver | Runs the application in the Java HotSpot VM server. |
| -Joption | Passes option to the JVM, where option is one of the options described on the reference page for the Java application launcher. For example, -J-Xms48m sets the startup memory to 48 MB. See Overview of Java Options in java. |
| -help | Displays a help message. |

* + - 1. The following options are forwarded to the debuggee process

|  |  |
| --- | --- |
| -v  -verbose[:class|gc|jni] | Turns on the verbose mode. |
| -Dname=value | Sets a system property. |
| -classpath dir | Lists directories separated by colons in which to look for classes. |
| -X option | A nonstandard target JVM option. |

### Operation

* + - 1. Start a JDB Session
         1. There are many ways to start a JDB session.
         2. The most frequently used way is to have the JDB launch a new JVM with the main class of the application to be debugged.

Do this by substituting the jdb command for the java command in the command line.

For example, if your application's main class is MyClass, then use the following command to debug it under the JDB:

jdb MyClass

When started this way, the jdb command calls a second JVM with the specified parameters, loads the specified class, and stops the JVM before executing that class's first instruction.

* + - * 1. Another way to use the jdb command is by attaching it to a JVM that's already running.

Syntax for starting a JVM to which the jdb command attaches when the JVM is running is as follows.

This loads in-process debugging libraries and specifies the kind of connection to be made.

java -agentlib:jdwp=transport=dt\_socket,server=y,suspend=n MyClass

You can then attach the jdb command to the JVM with the following command:

jdb -attach 8000

8000 is the address of the running JVM.

The MyClass argument isn't specified in the jdb command line in this case because the jdb command is connecting to an existing JVM instead of launching a new JVM.

* + - * 1. There are many other ways to connect the debugger to a JVM, and all of them are supported by the jdb command. The Java Platform Debugger Architecture has additional documentation on these connection options.
      1. Breakpoints
         1. Breakpoints can be set in the JDB at line numbers or at the first instruction of a method,
         2. for example:

The command stop at MyClass:22 sets a breakpoint at the first instruction for line 22 of the source file containing MyClass.

The command stop in java.lang.String.length sets a breakpoint at the beginning of the method java.lang.String.length.

The command stop in MyClass.<clinit> uses <clinit> to identify the static initialization code for MyClass.

* + - * 1. When a method is overloaded, you must also specify its argument types so that the proper method can be selected for a breakpoint.

For example, MyClass.myMethod(int,java.lang.String) or MyClass.myMethod().

* + - * 1. The clear command removes breakpoints using the following syntax: clear MyClass:45.

Using the clear or stop command with no argument displays a list of all breakpoints currently set.

* + - * 1. The cont command continues execution.
      1. Stepping
         1. The step command advances execution to the next line whether it's in the current stack frame or a called method.
         2. The next command advances execution to the next line in the current stack frame.
      2. Exceptions
         1. When an exception occurs for which there isn't a catch statement anywhere in the throwing thread's call stack, the JVM typically prints an exception trace and exits.
         2. When running under the JDB, however, control returns to the JDB at the offending throw.
         3. You can then use the jdb command to diagnose the cause of the exception.
         4. Use the catch command to cause the debugged application to stop at other thrown exceptions,

for example: catch java.io.FileNotFoundException or catch mypackage.BigTroubleException.

* + - * 1. Any exception that's an instance of the specified class or subclass stops the application at the point where the exception is thrown.
        2. The ignore command negates the effect of an earlier catch command.

The ignore command doesn't cause the debugged JVM to ignore specific exceptions, but only to ignore the debugger.

## jdeprscan

### Description

### Synopsis

### Example

## jdeps

### Description

### Synopsis

### Example

## jfr

### Description

### Synopsis

### Example

## jhsdb

### Description

### Synopsis

### Example

## jlink

### Description

### Synopsis

### Example

## jmod

### Description

### Synopsis

### Example

## jpackage

### Description

### Synopsis

### Example

## jrunscript

### Description

### Synopsis

### Example

## jshell

### Description

### Synopsis

### Example

## keytool

### Description

### Synopsis

### Exampl

## rmid

### Description

### Synopsis

### Example

## rmiregistry

### Description

* + - 1. The rmiregistry command creates and starts a remote object registry on the specified port on the current host.
      2. If the port is omitted, then the registry is started on port 1099.
      3. The rmiregistry command produces no output and is typically run in the background, for example:

rmiregistry &

* + - 1. A remote object registry is a bootstrap naming service that's used by RMI servers on the same host to bind remote objects to names.
      2. Clients on local and remote hosts can then look up remote objects and make remote method invocations.
      3. The registry is typically used to locate the first remote object on which an application needs to call methods.
      4. That object then provides application-specific support for finding other objects.
      5. The methods of the java.rmi.registry.LocateRegistry class are used to get a registry operating on the local host or local host and port.
      6. The URL-based methods of the java.rmi.Naming class operate on a registry and can be used to:
         1. Bind the specified name to a remote object
         2. Return an array of the names bound in the registry
         3. Return a reference, a stub, for the remote object associated with the specified name
         4. Rebind the specified name to a new remote object
         5. Destroy the binding for the specified name that's associated with a remote object

### Synopsis

rmiregistry [options] [port]

options

This represents the option for the rmiregistry command.

-Joption

Used with any Java option to pass the option following the -J (no spaces between the -J and the option) to the Java interpreter.

port

The number of a port on the current host at which to start the remote object registry.

## serialver

### Description

* + - 1. The serialver command returns the serialVersionUID for one or more classes in a form suitable for copying into an evolving class.
      2. When called with no arguments, the serialver command prints a usage line.
      3. Notes:
         1. The serialver command loads and initializes the specified classes in its virtual machine, and by default, it doesn't set a security manager.
         2. If the serialver command is to be run with untrusted classes, then a security manager can be set with the following option:

-J-Djava.security.manager

* + - * 1. When necessary, a security policy can be specified with the following option:

-J-Djava.security.policy=policy\_file

### Synopsis

serialver [options] [classnames]

options

This represents the command-line options for the serialver command.

-classpath path-files

Sets the search path for application classes and resources.

Separate classes and resources with a colon (:).

-Joption

Passes the specified option to the Java Virtual Machine, where option is one of the options described on the reference page for the Java application launcher.

For example, -J-Xms48m sets the startup memory to 48 MB.

classnames

The classes for which serialVersionUID is to be returned.

# OQL(Object Query Language)

## Description

* + - 1. OQL是用于查询Java堆的类SQL查询语言。OQL允许过滤/选择从Java堆中获取的信息。虽然HAT已经支持预定义的查询，例如“显示类X的所有实例”，但OQL增加了更多的灵活性。
      2. 本文整理eclipse的MAT工具和 JvisualVM的OQL语法

## BNF(Backus-Naur Form)

### Description

* + - * 1. <https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form>
        2. 以美国人巴科斯(Backus)和丹麦人诺尔(Naur)的名字命名的一种形式化的语法表示方法，用来描述语法的一种形式体系，是一种典型的元语言。又称巴科斯-诺尔形式(Backus-Naur form)。它不仅能严格地表示语法规则，而且所描述的语法是与上下文无关的。它具有语法简单，表示明确，便于语法分析和编译的特点。BNF表示语法规则的方式为：非终结符用尖括号括起。每条规则的左部是一个非终结符，右部是由非终结符和终结符组成的一个符号串，中间一般以“：：=”分开。具有相同左部的规则可以共用一个左部，各右部之间以直竖“|”隔开。

# Eclipse MAT OQL(1.11.0)

## Synopsis

### Document

https://help.eclipse.org/2020-12/index.jsp?topic=%2Forg.eclipse.mat.ui.help%2Freference%2Foqlsyntax.html&cp%3D61\_4\_2

https://wiki.eclipse.org/MemoryAnalyzer/OQL

https://www.ibm.com/support/knowledgecenter/SS3KLZ/com.ibm.java.diagnostics.memory.analyzer.doc/homepage/plugin-homepage-ma.html

* + - 1. 版本之间会有细节差异

### Syntax

SELECT [DISTINCT] [OBJECTS][AS RETAINED SET] \*  
FROM [OBJECTS][ INSTANCEOF ] <class name="name">  
[WHERE <filter-expression> ]  
[UNION otherOQL]

### SELECT

* + - 1. AS: use the AS keyword to name the columns

**SELECT** toString(s) **AS Value**,  
 s.@usedHeapSize **AS** "Shallow Size",  
 s.@retainedHeapSize **AS** "Retained Size"  
**FROM** java.lang.String s

* + - 1. DISTINCT: Use the DISTINCT keyword to only select unique objects

**SELECT DISTINCT** \* **FROM** OBJECTS 0,1,1,2

* + - 1. OBJECTS：Use the OBJECTS keyword if you want to process the text that follows the FROM keyword as objects instead of classes.

**SELECT** OBJECTS dominators(s) **FROM** java.lang.String s

* + - 1. AS RETAINED SET: use the AS RETAINED SET keyword to get the set of object retained by your selection

**SELECT   
 AS** RETAINED **SET** \*   
**FROM** java.lang.String

### FROM

* + - 1. INSTANCEOF: Use the INSTANCEOF keyword to include objects of sub-classes into the query

**SELECT** \* **FROM** INSTANCEOF java.lang.ref.Reference

* + - 1. OBJECTS：Use the OBJECTS keyword if you do not want to process the term as classes

**SELECT** \* **FROM** OBJECTS java.lang.String

* + - 1. By class name：

**SELECT** \* **FROM** java.lang.String

* + - 1. By a regular expression matching the class name

**SELECT** \* **FROM** "java\.lang\..\*"

* + - 1. By the object address of the class

**SELECT** \* **FROM** 0x2b7468c8

* + - 1. By the object addresses of more than one class

**SELECT** \* **FROM** 0x2b7468c8,0x2b74aee0

* + - 1. By a sub select

**SELECT** \*  
**FROM** (   
 **SELECT** \*  
 **FROM** java.lang.Class **c  
 WHERE   
 c** implements org.eclipse.mat.snapshot.model.IClass   
 )

* + - 1. By the object id of the class

**SELECT** \* **FROM** 20815

* + - 1. By the object ids of more than one class

**SELECT** \* **FROM** 20815,20975

### WHERE

* + - 1. 基本操作符：>=, <=, >, <, [ NOT ] LIKE, [ NOT ] IN, IMPLEMENTS (relational operations)
      2. 字符表达，Boolean, String, Integer, Long, Character, Float, Double and null literals
      3. 例子

**SELECT** \*   
**FROM** java.lang.String s  
**WHERE** ( s.count > 1000 ) = **true  
 OR** toString(s) = "monday"  
 **OR** dominators(s).size() = 0  
 **OR** s.@retainedHeapSize > 1024L  
 **OR** s.value != **null   
 AND** s.value.@valueArray.@length >= 1   
 **AND** s.value.@valueArray.get(0) = 'j'

**SELECT** \*  
**FROM** instanceof java.lang.Number s  
**WHERE** s.value > -1  
 **OR** s.value > -1L  
 **OR** s.value > 0.1  
 **OR** s.value > -0.1E-2F  
 **OR** s.value > 0.1D  
 **OR** s.value > -0.1E-2D  
 **OR** s.value > 0.1  
 **OR** s.value > -0.1E-2F  
 **OR** s.value > 0.1D  
 **OR** s.value > -0.1E-2D

**SELECT** \* **FROM** java.lang.String s **WHERE** s.count >= 100

**SELECT** \* **FROM** java.lang.String s **WHERE** toString(s) **LIKE** ".\*day"

**SELECT** \* **FROM** java.lang.String s **WHERE** s.value **NOT IN** dominators(s)

**SELECT** \* **FROM** java.lang.Class **c WHERE c** IMPLEMENTS org.eclipse.mat.snapshot.model.Iclass

**SELECT** \* **FROM** java.lang.String s **WHERE** toString(s) = "monday"

**SELECT** \*  
**FROM** java.lang.String s   
**WHERE** s.count > 100   
 **AND** s.@retainedHeapSize > s.@usedHeapSize

**SELECT** \*  
**FROM** java.lang.String s   
**WHERE** s.count > 1000   
 **OR** s.value.@length > 1000

### UNION

* + - 1. The UNION clause allows the results of two queries to be combined.
      2. The two queries must match in the number of columns in the select clause

**SELECT** s, s.value, s.hash   
**FROM** java.lang.String s   
**UNION**(  
**SELECT** b, b.value, "dummy"   
**FROM** java.lang.StringBuilder b

)

## Property Accessors

### Accessing fields of the heap object

* + - 1. Properties of heap objects are accessed using a simple dot notation

[<alias>.]<field>[.<field>...]

* + - * 1. alias:

can be defined in the FROM clause to identify the current object

i.e. row in the SQL analogy, on which the OQL statement operates

Without alias,the field is assumed to be one of the fields of the current object

* + - * 1. fields: are attributes of the java objects in the heap dump

**SELECT** s.count, s.value **FROM** java.lang.String s

### Calling Java Bean properties：

[<alias>.]@<attribute> ...

* + - 1. Using the @ symbol, OQL accesses attributes of the underlying Java object used by Memory Analyzer to repesent object in the heap dump
      2. The attributes are resolved via Bean Introspection
      3. The following table lists some commonly used Java attributes

**SELECT** s.@usedHeapSize, s.@retainedHeapSize **FROM** java.lang.String s

| **Any heap object** | **IObject** | ObjectId | **ID of the snapshot object** |
| --- | --- | --- | --- |
| 所有Object | | objectAddress | address of the snapshot object |
| class | Java class of this object |
| clazz | IClass of this object. See also classof(object) |
| usedHeapSize | shallow heap size |
| retainedHeapSize | retained heap size |
| displayName | display name |
| Class object | IClass | classLoaderId | ID of the class loader |
| Any array | IArray | length | length of the array |
| Primitive array | IPrimitiveArray | valueArray | the values in the array |
| Reference array | IObjectArray | referenceArray | the objects in the array (the addresses of the objects as long values). Access a particular element by using the get() method, then convert to an object by using the OBJECTS keyword. |

### Calling Java methods

[<alias>.]@<method>([<expression>,<expression>]) ...

The call is executed via reflection

Example SELECT s.toString(s) FROM java.lang.String s

Java对象底层的方法：

| **Heap object** | **IObject** | **Method** | **Return object** |
| --- | --- | --- | --- |
| ${snapshot} | ISnapshot | getClasses() | A collection of all classes |
| getClassesByName(String name, boolean includeSubClasses) | A collection of classes |
| Class object | IClass | hasSuperClass() | The result is true if the class has a super class |
| isArrayType() | The result is true if the class is an array type |
| Any heap object | IObject | getObjectAddress() | The address of a snapshot object as a long integer |
| Primitive array | IPrimitiveArray | getValueAt(int index) | A value from the array |
| Java primitive array  Java object array  Java list | [] or List | get(index) | A value from the array or list |

### Array Access

* + - 1. Mat 1.3 or later allow direct array style access of primitive arrays an object arrays from the snapshot and java arrays and java Lists obtained from reflective method calls
      2. The index is a zero-based integer
      3. If the array is null or the index is out of range then the result is null
      4. Mat 1.4 or later allows array range access as well using the notation [index1:index2], where index1 and index2 are inclusive
      5. If the values are negative
         1. they are treated as indexing from the end of the array
         2. So -1 means the last entry
         3. This mean the whole array can be accessed as a list as [0:-1]
      6. Example
         1. Reading values from primitive arrays(From the heap dump)

This method is for mat 1.3+

**SELECT** s[2] **FROM int**[] s **WHERE** (s.@length > 2)

This method is for all versions of mat.

**SELECT** s.getValueAt(2) **FROM int**[] s **WHERE** (s.@length > 2)

* + - * 1. Reading object from objcet arrays

Mat 1.3+

**SELECT** s[2] **FROM** java.lang.Object[] s **WHERE** (s.@length > 2)

Mat 1.3+

**SELECT** OBJECTS s[2] **FROM** java.lang.Object[] s

Mat 1.1+

**SELECT** OBJECTS s.@referenceArray.get(2) **FROM** java.lang.Object[] s **WHERE** (s.@length > 2)

Mat 1.1+

**SELECT** OBJECTS s.getReferenceArray(2,1) **FROM** java.lang.Object[] s **WHERE** (s.@length > 2)

* + - * 1. Reading from Java arrays(MAT internal objects)

Mat 1.3+

**SELECT** s.@GCRoots[2] **FROM** OBJECTS ${snapshot} s

All mat

**SELECT** s.get(2) **FROM** OBJECTS ${snapshot} s **WHERE** s.@GCRoots.@length > 2

* + - * 1. Read from Java Lists(MAT internal objects)

MAT 1.3+

**SELECT** s.@GCRoots.subList(1,3)[1] **FROM** OBJECTS ${snapshot} s

All mat

**SELECT** s.@GCRoots.subList(1,3).get(1) **FROM** OBJECTS ${snapshot} s

* + - * 1. Reading subarrays

MAT 1.4+

**SELECT** s, s.count, s.offset, s.value[s.offset],  
 s.value[s.offset:((s.offset + s.count) - 1)],  
 s.value[s.offset:((s.offset + 0) - 1)],  
 s.value[0:-1].subList(s.offset,(s.offset + 0)),  
 s.value[s.offset:-1].subList(0,s.count)  
**FROM** java.lang.String

* + - * 1. Collection access

Many of the standard collections classes are well known by MAT.

The collection queries allow analysis of lists,sets,queues,deques and maps

This access is extended to OQL so if the collection queries work with a particular collection or map then so does OQL

**SELECT** a[0] **FROM** java.util.ArrayList a

**SELECT** a[0:-1] **FROM** java.util.ArrayList a

**SELECT** h[0].@key, h[0].@value **FROM** java.util.HashMap h

### Built-in OQL function

|  |  |
| --- | --- |
| function | descript |
| toHex(number) | Print the number as hexadcimal |
| toString(object) | Return the value of an object. |
| dominators(object) | The objects immediately dominated by the object |
| outbounds(object) | Outbound referrer |
| inbounds(object) | Inbound referrer |
| classof(object) | The class of the current object |
| dominatorof(object) | The immediate dominator.  -1 if none |
| eval(expression) | Experimental in mat 1.4 or later  Evaluates the argument and return it.  Could be useful to allow array/method access to the result of a sub-select or expression |

## Simulated in OQL

* + - 1. LIMIT and OFFSET

**SELECT** eval((  
 **SELECT** \*   
 **FROM** OBJECTS(  
 **SELECT** s,   
 s.value **AS** val   
 **FROM** java.lang.String s   
 )v  
 ))[3]   
**FROM** OBJECTS 0

**SELECT** z.s  
**FROM** OBJECTS(  
 eval((  
 **SELECT** s   
 **FROM** "java.lang.String" s   
 ))[10:29]   
 ) z

* + - 1. GROUP BY

**SELECT** s.sz **AS Size**,  
 (  
 **SELECT** OBJECTS m  
 **FROM** java.util.HashMap m  
 **WHERE** (  
 m[0:-1].size() = s.sz  
 )  
 ) **AS** Maps  
**FROM** OBJECTS (   
 **SELECT   
 DISTINCT** h[0:-1].size() **AS** sz   
 **FROM** java.util.HashMap h   
 ) s

* + - * 1. Grouping by number of inbounds

**SELECT** s.sz **AS Size**,   
 (  
 **SELECT** OBJECTS m   
 **FROM** INSTANCEOF java.lang.Object m   
 **WHERE** (  
 inbounds(m).@length = s.sz  
 )  
 ) **AS** Objects  
**FROM** OBJECTS (  
 **SELECT   
 DISTINCT** inbounds(h).@length **AS** sz   
 **FROM** INSTANCEOF java.lang.Object h  
 ) s

* + - 1. COUNT

**SELECT** z.size **AS Size**,  
 z.maps **AS** Maps,  
 z.maps.@length **AS** "Count",  
 z.maps[0:-1].size() **AS** "Count (another way)"  
**FROM** OBJECTS (  
 eval((  
 **SELECT** s.sz **AS size**,  
 (  
 **SELECT** OBJECTS m  
 **FROM** java.util.HashMap m  
 **WHERE** (  
 m[0:-1].size() = s.sz  
 )  
 ) **AS** maps  
 **FROM** OBJECTS (  
 **SELECT  
 DISTINCT** h[0:-1].size() **AS** sz  
 **FROM** java.util.HashMap h  
 ) s  
 ))  
 ) z



**SELECT** z.size **AS Size**,  
 z.objects **AS** Objects,  
 z.objects.@length **AS** "Count",  
 z.objects[0:-1].size() **AS** "Count (another way)"  
**FROM** OBJECTS (  
 eval((  
 **SELECT** s.sz **AS size**,  
 (  
 **SELECT** OBJECTS m  
 **FROM** INSTANCEOF java.lang.Object m  
 **WHERE** (  
 inbounds(m).@length = s.sz  
 )  
 ) **AS** objects  
 **FROM** OBJECTS (  
 **SELECT  
 DISTINCT** inbounds(h).@length **AS** sz  
 **FROM** INSTANCEOF java.lang.Object h  
 ) s  
 ))  
 ) z

* + - 1. CROSS JOIN

**SELECT** z.i **AS Integer**,   
 z.i.value **AS** "Integer value",   
 z.lv.l **AS** Long,   
 z.lv.l.value **as** "Long value"  
**FROM** OBJECTS (   
 **SELECT** i,   
 (  
 **SELECT** l   
 **FROM** java.lang.Long l   
 ) **AS** lv   
 **FROM** java.lang.Integer i  
 ) z

* + - 1. LEFT JOIN

**SELECT** z.i **AS Integer**,   
 z.i.value **AS** "Integer value",   
 z.lv.l **AS** Long,   
 z.lv.l.value **as** "Long value"  
**FROM** OBJECTS (  
 **SELECT** i,   
 (  
 **SELECT** l  
 **FROM** java.lang.Long l   
 **WHERE** (  
 l.value = i.value  
 )  
 ) **AS** lv   
 **FROM** java.lang.Integer i  
 ) z

* + - 1. INNER JOIN

**SELECT** z.i **AS Integer**,   
 z.i.value **AS** "Integer value",   
 z.lv.l **AS** Long,   
 z.lv.l.value **as** "Long value"  
**FROM** OBJECTS (   
 **SELECT** i,   
 (  
 **SELECT** l   
 **FROM** java.lang.Long l   
 **WHERE** (  
 l.value = i.value  
 )  
 ) **AS** lv   
 **FROM** java.lang.Integer i   
 ) z  
**WHERE** (  
 z.lv != **null** )

**SELECT** z.iv.i **AS Integer**,   
 z.iv.i.value **AS** "Integer value",   
 z.l **AS** Long,   
 z.l.value **as** "Long value"  
**FROM** OBJECTS (   
 **SELECT** (  
 **SELECT** i  
 **FROM** java.lang.Integer i   
 **WHERE** (  
 i.value = l.value  
 )  
 ) **AS** iv,   
 l   
 **FROM** java.lang.Long l   
 ) z  
**WHERE** (  
 z.iv != **null** )

* + - 1. RIGHT JOIN

**SELECT** z.iv.i **AS Integer**,   
 z.iv.i.value **AS** "Integer value",   
 z.l **AS** Long,   
 z.l.value **as** "Long value"  
**FROM** OBJECTS (  
 **SELECT** (  
 **SELECT** i  
 **FROM** java.lang.Integer i   
 **WHERE** (  
 i.value = l.value  
 )  
 ) **AS** iv,   
 l   
 **FROM** java.lang.Long l   
 ) z

* + - 1. FULL OUTER JOIN

**SELECT** z.i **AS Integer**,   
 z.i.value **AS** "Integer value",   
 z.lv.l **AS** Long,   
 z.lv.l.value **as** "Long value"  
**FROM** OBJECTS (  
 **SELECT** i,   
 (  
 **SELECT** l   
 **FROM** java.lang.Long l   
 **WHERE** (  
 l.value = i.value  
 )  
 ) **AS** lv   
 **FROM** java.lang.Integer i  
 ) z  
**UNION** (  
 **SELECT** z.iv.i **AS Integer**,   
 z.iv.i.value **AS** "Integer value",   
 z.l **AS** Long,   
 z.l.value **as** "Long value"  
 **FROM** OBJECTS (  
 **SELECT** (  
 **SELECT** i   
 **FROM** java.lang.Integer i   
 **WHERE** (  
 i.value = l.value  
 )  
 ) **AS** iv,   
 l   
 **FROM** java.lang.Long l   
 ) z  
 **WHERE** (  
 z.iv = **null** )  
)

## More Example

**SELECT** \* **FROM** ${snapshot}.getClassesByName("java.lang.ref.Reference", **true**)

**SELECT** toString(s) **AS Value**,   
 s.@usedHeapSize **AS** "Shallow Size",   
 s.@retainedHeapSize **AS** "Retained Size"   
**FROM** java.lang.String s

**SELECT AS** RETAINED **SET** \* **FROM** java.lang.String

**SELECT DISTINCT** OBJECTS classof(s) **FROM** java.lang.String s

**SELECT DISTINCT** \* **FROM** OBJECTS 0,1,1,2

**SELECT** OBJECTS dominators(s) **FROM** java.lang.String s

**SELECT** toString(s), s.count, s.value **FROM** java.lang.String s

**SELECT** s.@objectId,   
 (s.@objectId \* 2),   
 ("The object ID is " + @objectId)  
**FROM** OBJECTS 0,1,1,2 s

## BNF for the Object Query Language

https://help.eclipse.org/2020-12/index.jsp?topic=%2Forg.eclipse.mat.ui.help%2Freference%2Foqlsyntax.html&cp%3D61\_4\_2

# JVisualMV OQL

## Synopsis

### Document

https://blogs.oracle.com/sundararajan/querying-java-heap-with-oql http://cr.openjdk.java.net/~sundar/8022483/webrev.01/raw\_files/new/src/share/classes/com/sun/tools/hat/resources/oqlhelp.html

https://visualvm.github.io/

https://visualvm.github.io/documentation.html

### Syntax

**select** <JavaScript **expression to select**>  
[  
**from** [instanceof] < class **name**><identifier>  
[  
**where** <JavaScript boolean **expression to** filter>  
]  
]

* + - 1. 支持JavaScript表达式
      2. 关键字全小写
      3. 关系符：>=,<=,>,<,[NOT] LIKE,[NOT] IN,IMPLEMENTS,=,!=,AND,OR
      4. jdk1.9开始，不再绑定

## Built-in Object——heap

### Introduction

|  |  |
| --- | --- |
| 对象 | 说明 |
| heap.forEachClass(callback) | 遍历Class  对每一个Class对象执行一个回调操作。其中 callback 为 Javascript 函数 |
| heap.forEachObject (callback, clazz, includeSubtypes) | 遍历Object  clazz:指定实例对象，默认为Java.lang.Object  includeSubtypes：是否包含子类，默认true |
| heap.findClass(className) | 查找给定名称的Java类, 生成的Class对象具有以下属性:  name - name of the class.  superclass - Class object for super class (or null if java.lang.Object).  statics - name, value pairs for static fields of the Class.  fields - array of field objects. field object has name, signature properties.  loader - ClassLoader object that loaded this class.  signers - signers that signed this class.  protectionDomain - protection domain to which this class belongs  生成的Class对象同时具有一下方法：  isSubclassOf - tests whether given class is direct or indirect subclass of this class or not.  isSuperclassOf - tests whether given Class is direct or indirect superclass of this class or not.  subclasses - returns array of direct and indirect subclasses.  superclasses - returns array of direct and indirect superclasses. |
| head.findObject(objID) | 根据对象ID找对象 |
| heap.classes() | 返回堆快照中所有的类的枚举 |
| heap.objects(clazz, [includeSubtypes], [filter]) | 返回堆快照中所有的对象的枚举  clazz:指定类名称，默认java.lang.Object  includeSubtypes：是否包含子类，true  filter:过滤规则,js函数或过滤规则 |
| head.livepaths(obj,flag) | Return an array of paths by which a given object is alive.  This method accepts optional second parameter that is boolean flag  This flag tells whether to include paths with weak reference(s) or not  By default,paths with weak reference(s) are not included  Each element of this array itself is another array  The later array is contains an objects that are in the 'reference chain' of the path |
| heap.roots() | Return an Enumeration of Roots of the haep  Each Root Object has the following properties:  id - String id of the object that is referred by this root  type - descriptive type of Root (JNI Global, JNI Local, Java Static etc)  description - String description of the Root  referrer - Thread Object or Class object that is responsible for this root or null |
| heap.finalizables() | Returns an enumeration of java object that are pending to finalized |

### Example

**select** heap.findClass("java.lang.System").statics.props

**select** heap.findClass("java.lang.String").fields.length

**select** heap.findObject("0xf3800b58")

**select** filter(heap.classes(), "/java.net./.test(it.name)")

## Functions on individual objects

### Introduction

|  |  |
| --- | --- |
| 函数 | 说明 |
| allocTrace(objName) | This returns allocation site trace of a given Java object if available  allocTrace returns array of frame objects.  Each frame object has following properties:  className - name of the Java class whose method is running in the frame.  methodName - name of the Java method running in the frame.  methodSignature - signature of the Java method running in the frame.  sourceFileName - name of source file of the Java class running in the frame.  lineNumber - source line number within the method. |
| classof(objname) | Return Class object of a given Java Object.  The result object supports the following properties:  name - name of the class.  superclass - Class object for super class (or null if java.lang.Object).  statics - name, value pairs for static fields of the Class.  fields - array of field objects. Field objects have name, signature properties.  loader - ClassLoader object that loaded this class.  signers - signers that signed this class.  protectionDomain - protection domain to which this class belongs.  Class objects have the following methods：  isSubclassOf() - tests whether given class is direct or indirect subclass of this class or not.  isSuperclassOf() - tests whether given Class is direct or indirect superclass of this class or not.  subclasses() - returns array of direct and indirect subclasses.  superclasses() - returns array of direct and indirect superclasses. |
| forEachReferrer(callback,obj) | calls a callback function for each referrer of a given java object |
| identical(o1,o2) | Returns whether two given Java objects are identical or not |
| objectid(objName) | Returns String id of a given Java object  This id can be passed to head.findObject() and may also be used to compare objects for identity |
| reachables(obj[,omit]) | Returns an array of Java objects that are transitively referred from the given java object  Optionally accepts a second parameter that is comma separated field names to be excluded from reachability computation.  Fields are written in class\_name.field\_name pattern |
| referrers(obj) | Returns an enumeration of java object that hold reference to a give java object |
| referees(obj) | Returns an array of java objects to which the given java object directly refers to |
| refers(o1,o2) | Returns whether first java object refers to second java object or not |
| root(obj) | If given Object is member of root set of objects, this function returns a descriptive Root object describing why it is so.  if given Object is not a root , then this function returns null |
| sizeof(obj) | Return size of given java object in bytes |
| toHtml(obj) | Return HTML string for the given java object.  Note that this is called automatically for objects selected by select expression  But, it may be useful to print more complex output. |
| rsizeof(obj) |  |

### Example



**select** classof(o).name **from** instanceof java.lang.ref.Reference o



**select** heap.findClass("java.io.InputStream").subclasses()



**select** heap.findClass("java.io.BufferedInputStream").superclasses()

**select** identical(  
 heap.findClass("Foo").statics.bar,   
 heap.findClass("AnotherClass").statics.bar  
 )

**select** objectid(o) **from** java.lang.Object o

**select** reachables(p) **from** java.util.Properties p

**select** reachables(u, 'java.net.URL.handler') **from** java.net.URL u

**select count**(referrers(o)) **from** java.lang.Object o

**select** referrers(f) **from** java.io.File f

**select** u **from** java.net.URL u **where count**(referrers(u)) > 2

**select** referees(heap.findClass("java.io.File"))

**select** sizeof(o) **from** [I o

**select** "<b>" + toHtml(o) + "</b>" **from** java.lang.Object o

## array/iterator/enumeration manipulation functions

### Introduction

|  |  |
| --- | --- |
| 函数 | 说明 |
| concat(objSet1,ObjSet2) | Concatenates two arrays or enumerations |
| contains(objSet, booleanExpression) | Returns whether the given array/enumeration contains an element the given boolean expression specified in code  The code evaluated can refer to the following built-in variable:  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated |
| count(objSet, booleanExpression) | count function returns the count of elements of the input array/enumeration that satisfy the given boolean expression  The boolean expression code can refer to the following built-in variables:  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated |
| filter(objSet, booleanExpression) | filter function returns an array/enumeration that contains elements of the input array/enumeration that satisfy the given boolean expression.  The boolean expression code can refer to the following built-in variables:  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated |
| length(objSet) | length function returns number of elements of an array/enumeration |
| map(objSet,transerRule) | map function returns an array/enumeration of values created by repeatedly calling code on each element of input array/enumeration  Transforms the given array/enumeration by evaluation given code on each element  The code evaluated can refer to the following built-in variable:  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated  result -> result array/enumeration |
| max(setObj,[express]) | returns the maximum element of given array/enumeration  Optionally accept code expression to compare elements of the arary  By default numerical comparison is used  The comparison expression can use the following built-in variables:  lhs -> left side element for comparison  rhs -> right side element for comparison |
| min(setObj,[express]) | returns the minimum element of the given array/enumeration  Optionally accepts code expression to compare elements of the array  By default numerical comparison is used  The comparison expression can use the following built-in variable:  lhs -> left side element for comparison  rhs -> right side element for comparison |
| sort(setObj,[express]) | sort given array/enumeration  Optional accepts code expression to compare elements of the array  by default numerical comparison is used  The comparison expression can use the following built-in variable:  lhs -> left side element for comparison  rhs -> right side element for comparison |
| sum(setObj,[express]) | This function returns the sum of all the elements of the given input array/enumeration  Optionally accepts an expression as second param.  This is used to map the input elements before summing those. |
| toArray(objSet) | this function return an array/enumeration containing unique elements of the given input array/enumeration |
| unique(objSet) | this function returns an array/enumeration containing unique elements of the given input array/enumeration |
| top(array/enumeration,[expression],to) | return top N elements of the given array/enumeration |

### Example



**select** p   
**from** java.util.Properties p  
**where   
 contains**(  
 referrers(p),   
 "classof(it).name == 'java.lang.Class'"  
 )



**select count**(heap.classes(), "/java.io./.test(it.name)")



**select** filter(heap.classes(), "/java.io./.test(it.name)")



**select** filter(  
 referrers(u),   
 "! /java.net./.test(classof(it).name)"  
 )  
**from** java.net.URL u



**select  
 map**(  
 heap.findClass("java.io.File").statics,   
 "index + '=' + toHtml(it)"  
 )



**select max**(**map**(heap.objects('java.lang.String', **false**), 'it.value.length'))



**select max**(heap.objects('java.lang.String'), 'lhs.value.length > rhs.value.length')



**select min**(**map**(heap.objects('java.util.Vector', **false**), 'it.elementData.length'))



**select   
 min**(  
 heap.objects('java.util.Vector'),   
 'lhs.elementData.length < rhs.elementData.length'  
 )



**select** sort(heap.objects('[C'), 'sizeof(lhs) - sizeof(rhs)')



**select   
 map**(  
 sort(  
 heap.objects('[C'),  
 'sizeof(lhs) - sizeof(rhs)'  
 ),   
 '{ size: sizeof(it), obj: it }'  
 )



**select sum**(**map**(reachables(p), 'sizeof(it)')) **from** java.util.Properties p



**select sum**(reachables(p), 'sizeof(it)') **from** java.util.Properties p



**select count**(**unique**(**map**(heap.objects('java.lang.String'), 'it.value')))



**select count**(heap.objects('java.lang.String'))



**select   
 map**(  
 top(  
 heap.objects('java.lang.String'),   
 'rhs.count - lhs.count',   
 5  
 ),   
 '{ length: it.count, obj: it }'  
 )

## More examples

**select map**(  
 heap.objects('java.lang.ClassLoader'),  
 **function** (it) {  
 var res = '';  
 while (it != **null**) {  
 res += toHtml(it) + "->";  
 it = it.parent;  
 }  
 res += "null";  
 **return** res + "<br>";  
 }  
 )

**select** {  
 obj: f.referent,  
 **size** : **sum** (**map** (reachables(f.referent), "sizeof(it)"))  
 }  
**from** java.lang.ref.Finalizer f  
**where** f.referent != **null**

**select** s **from** java.lang.String s **where** s.value.length >= 100

**select** a **from** [I a **where** a.length >= 256

**select**

s.value.toString()   
**from** java.lang.String s  
**where** /java/.test(s.value.toString())

**select** file.path.value.toString() **from** java.io.File file

**select** classof(cl).name **from** instanceof java.lang.ClassLoader cl

**select** o **from** instanceof 0xd404b198 o

**select** s **from** java.lang.String s **where** s.value.length >= 100

**select** a **from** [I a **where** a.length >= 256

**select** "<b>" + toHtml(o) + "</b>"   
**from** java.lang.Object o



* + - 1. 显示所有File对象的文件路径:

select file.path.value.toString() from java.io.File file

* + - 1. 选取所有的ClassLoader，包括子类:

select classof(cl).name from instanceof java.lang.ClassLoader cl

* + - 1. 由给定id字符串标识的Class的实例

select o from instanceof 0x741012748 o(0x741012748是类的ID)

* + - 1. 表示两位数整数的字符串:

select {instance: s, content: s.toString()} from java.lang.String s where /^\d{2}$/(s.toString())

* + - 1. 打印class load以及对应的class数量

1. **select** map(sort(map(heap.objects('java.lang.ClassLoader'),
2. '{ loader: it, count: it.classes.elementCount }'), 'lhs.count < rhs.count'),
3. 'toHtml(it) + "<br>"')
   * + 1. 打印所有系统参数
4. **select** map(filter(heap.findClass('java.lang.System').statics.props.**table**, 'it != null'),
5. **function** (it) {
6. var res = "";
7. while (it != null) {
8. res += it.**key**.value.toString() + '=' +
9. it.value.value.toString() + '<br>';
10. it = it.**next**;
11. }
12. **return** res;
13. });
    * + 1. 打印ClassLoader的实例父-子链
14. **select** map(heap.objects('java.lang.ClassLoader'),
15. **function** (it) {
16. var res = '';
17. while (it != null) {
18. res += toHtml(it) + "->";
19. it = it.parent;
20. }
21. res += "null";
22. **return** res + "<br>";
23. })

查看线程状态

**select map**(heap.objects('java.lang.Thread'),   
 **function**(t){  
 var status = t.threadStatus;  
 if ((status & 4) != 0) {  
 **return** 'RUNNABLE';  
 } **else** if ((status & 1024) != 0) {  
 **return** 'BLOCKED';  
 } **else** if ((status & 16) != 0) {  
 **return** 'WAITING';  
 } **else** if ((status & 32) != 0) {  
 **return** 'TIMED\_WAITING';  
 } **else** if ((status & 2) != 0) {  
 **return** 'TERMINATED';  
 } **else** {  
 **return** (status & 1) == 0 ? 'NEW' : 'RUNNABLE';  
 }  
 }   
)

# 可视化工具列表

## Visual VM

## JConsole

## Mission Control

## JHSDB

## Eclipse MAT