

JVM诊断工具

[零 Reference 5](#_Toc65853988)

[一 JDK Analysis Tool 5](#_Toc65853989)

[1 jps 5](#_Toc65853990)

[1.1 Description 5](#_Toc65853991)

[1.2 Synopsis 5](#_Toc65853992)

[1.3 Output Format of the jps command 6](#_Toc65853993)

[1.4 Example 6](#_Toc65853994)

[2 jstat 7](#_Toc65853995)

[2.1 Description 7](#_Toc65853996)

[2.2 Synopsis 7](#_Toc65853997)

[2.3 Virtual Machine Identifier 7](#_Toc65853998)

[2.4 Output Options and Output 8](#_Toc65853999)

[2.5 Example 13](#_Toc65854000)

[3 jinfo 14](#_Toc65854001)

[3.1 Description 14](#_Toc65854002)

[3.2 Synopsis 15](#_Toc65854003)

[4 jmap 15](#_Toc65854004)

[4.1 说明 15](#_Toc65854005)

[4.2 文档 15](#_Toc65854006)

[4.3 命令 15](#_Toc65854007)

[4.4 参数 15](#_Toc65854008)

[5 jhat 16](#_Toc65854009)

[6 jstack 16](#_Toc65854010)

[6.1 说明 16](#_Toc65854011)

[6.2 文档 16](#_Toc65854012)

[6.3 命令 16](#_Toc65854013)

[6.4 参数 16](#_Toc65854014)

[7 jstatd 16](#_Toc65854015)

[7.1 说明 16](#_Toc65854016)

[7.2 文档 16](#_Toc65854017)

[7.3 命令 16](#_Toc65854018)

[7.4 参数 16](#_Toc65854019)

[8 jcmd 16](#_Toc65854020)

[8.1 说明 16](#_Toc65854021)

[8.2 文档 16](#_Toc65854022)

[8.3 命令 17](#_Toc65854023)

[8.4 参数 17](#_Toc65854024)

[二 JVM常用执行指令 17](#_Toc65854025)

[1 java 17](#_Toc65854026)

[1.1 说明 17](#_Toc65854027)

[1.2 文档 17](#_Toc65854028)

[1.3 基础指令 17](#_Toc65854029)

[1.4 参数 17](#_Toc65854030)

[2 javac 24](#_Toc65854031)

[2.1 文档 24](#_Toc65854032)

[2.2 基础指令 24](#_Toc65854033)

[2.3 主要参数 24](#_Toc65854034)

[3 jar 27](#_Toc65854035)

[3.1 Description 27](#_Toc65854036)

[3.2 Synopsis 27](#_Toc65854037)

[3.3 Options 27](#_Toc65854038)

[3.4 Example 28](#_Toc65854039)

[4 javap 29](#_Toc65854040)

[4.1 说明 29](#_Toc65854041)

[4.2 基础指令 29](#_Toc65854042)

[4.3 主要参数 29](#_Toc65854043)

[三 Other 30](#_Toc65854044)

[1 jaotc 30](#_Toc65854045)

[1.1 Description 30](#_Toc65854046)

[1.2 Synopsis 30](#_Toc65854047)

[1.3 Options 30](#_Toc65854048)

[1.4 Example 31](#_Toc65854049)

[1.5 What is Ahead of Time Compilation: 31](#_Toc65854050)

[2 jarsigner 31](#_Toc65854051)

[2.1 Description 31](#_Toc65854052)

[2.2 Synopsis 32](#_Toc65854053)

[2.3 Options 33](#_Toc65854054)

[2.4 Example 36](#_Toc65854055)

[3 javadoc 37](#_Toc65854056)

[3.1 Description 37](#_Toc65854057)

[3.2 Synopsis 37](#_Toc65854058)

[3.3 Example 37](#_Toc65854059)

[4 jconsole 37](#_Toc65854060)

[4.1 Description 37](#_Toc65854061)

[4.2 Synopsis 37](#_Toc65854062)

[4.3 Example 37](#_Toc65854063)

[5 jdb 37](#_Toc65854064)

[5.1 Description 37](#_Toc65854065)

[5.2 Synopsis 37](#_Toc65854066)

[5.3 Example 37](#_Toc65854067)

[6 jdeprscan 37](#_Toc65854068)

[6.1 Description 37](#_Toc65854069)

[6.2 Synopsis 37](#_Toc65854070)

[6.3 Example 37](#_Toc65854071)

[7 jdeps 38](#_Toc65854072)

[7.1 Description 38](#_Toc65854073)

[7.2 Synopsis 38](#_Toc65854074)

[7.3 Example 38](#_Toc65854075)

[8 jfr 38](#_Toc65854076)

[8.1 Description 38](#_Toc65854077)

[8.2 Synopsis 38](#_Toc65854078)

[8.3 Example 38](#_Toc65854079)

[9 jhsdb 38](#_Toc65854080)

[9.1 Description 38](#_Toc65854081)

[9.2 Synopsis 38](#_Toc65854082)

[9.3 Example 38](#_Toc65854083)

[10 jlink 38](#_Toc65854084)

[10.1 Description 38](#_Toc65854085)

[10.2 Synopsis 38](#_Toc65854086)

[10.3 Example 38](#_Toc65854087)

[11 jmod 38](#_Toc65854088)

[11.1 Description 38](#_Toc65854089)

[11.2 Synopsis 38](#_Toc65854090)

[11.3 Example 38](#_Toc65854091)

[12 jpackage 38](#_Toc65854092)

[12.1 Description 38](#_Toc65854093)

[12.2 Synopsis 38](#_Toc65854094)

[12.3 Example 38](#_Toc65854095)

[13 jrunscript 38](#_Toc65854096)

[13.1 Description 38](#_Toc65854097)

[13.2 Synopsis 38](#_Toc65854098)

[13.3 Example 38](#_Toc65854099)

[14 jshell 38](#_Toc65854100)

[14.1 Description 38](#_Toc65854101)

[14.2 Synopsis 38](#_Toc65854102)

[14.3 Example 38](#_Toc65854103)

[15 keytool 38](#_Toc65854104)

[15.1 Description 39](#_Toc65854105)

[15.2 Synopsis 39](#_Toc65854106)

[15.3 Exampl 39](#_Toc65854107)

[16 rmid 39](#_Toc65854108)

[16.1 Description 39](#_Toc65854109)

[16.2 Synopsis 39](#_Toc65854110)

[16.3 Example 39](#_Toc65854111)

[17 rmiregistry 39](#_Toc65854112)

[17.1 Description 39](#_Toc65854113)

[17.2 Synopsis 39](#_Toc65854114)

[17.3 Example 39](#_Toc65854115)

[18 serialver 39](#_Toc65854116)

[18.1 Description 39](#_Toc65854117)

[18.2 Synopsis 39](#_Toc65854118)

[18.3 Example 39](#_Toc65854119)

[四 OQL(Object Query Language) 39](#_Toc65854120)

[1 Description 39](#_Toc65854121)

[2 BNF(Backus-Naur Form) 39](#_Toc65854122)

[2.1 Description 39](#_Toc65854123)

[五 Eclipse MAT OQL(1.11.0) 39](#_Toc65854124)

[1 总揽 39](#_Toc65854125)

[1.1 简介 39](#_Toc65854126)

[1.2 SELECT 40](#_Toc65854127)

[1.3 FROM 40](#_Toc65854128)

[1.4 WHERE 41](#_Toc65854129)

[1.5 UNION 42](#_Toc65854130)

[2 Property Accessors 42](#_Toc65854131)

[2.1 Accessing fields of the heap object 42](#_Toc65854132)

[2.2 Calling Java Bean properties： 42](#_Toc65854133)

[2.3 Calling Java methods 43](#_Toc65854134)

[2.4 Array Access 43](#_Toc65854135)

[2.5 Built-in OQL function 44](#_Toc65854136)

[3 Simulated in OQL 45](#_Toc65854137)

[4 More Example 50](#_Toc65854138)

[5 BNF for the Object Query Language 50](#_Toc65854139)

[六 JVisualMV OQL 50](#_Toc65854140)

[1.1 概述 50](#_Toc65854141)

[2 Built-in Object——heap 51](#_Toc65854142)

[2.1 Introduction 51](#_Toc65854143)

[2.2 Example 51](#_Toc65854144)

[3 Functions on individual objects 52](#_Toc65854145)

[3.1 Introduction 52](#_Toc65854146)

[3.2 Example 52](#_Toc65854147)

[4 array/iterator/enumeration manipulation functions 53](#_Toc65854148)

[4.1 Introduction 53](#_Toc65854149)

[4.2 Example 54](#_Toc65854150)

[5 More examples 55](#_Toc65854151)

[七 可视化工具列表 57](#_Toc65854152)

[1 Visual VM 57](#_Toc65854153)

[2 JConsole 57](#_Toc65854154)

[3 Mission Control 57](#_Toc65854155)

[4 JHSDB 57](#_Toc65854156)

[5 Eclipse MAT 57](#_Toc65854157)

# 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 JDK15

# 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

### 说明

导出线程栈信息，进行死锁检查

### 命令

### 参数

|  |  |
| --- | --- |
| -F | 强制打印栈信息 |
| -l | 长列表，打印锁的附加信息 |
| -m | 打印Java 和 native C/C++ frames的所有栈信息 |
| -h ｜-help | 打印帮助信息 |

## jstatd

### 说明

启用远程监控，需要配置java的安全策略，并保存于jstatd.all.policy文件中

命令

jstatd J-Djava.security.policy=jstatd.all.policy [ options ]

### 参数

|  |  |
| --- | --- |
| -nr | 找不到RMI注册表时，不尝试创建 |
| -p | 指定端口 |
| -n | RMI名称，默认JstatRemoteHost，如果本地有多个jstatd服务，需要保证唯一 |

## jcmd

### 说明

* + - 1. a将诊断命令请求发送到本地正在运行的JVM，用来导出堆、查看Java进程、导出线程信息、执行GC、还可以进行采样分析
      2. b、执行者必须跟JVM是同一用户和用户组

### 命令

jcmd <pid | main-class> <command ... | PerfCounter.print | option>

### 参数

|  |  |
| --- | --- |
| -l | 列出所有JVM |
| -h ｜-help | 列出JVM支持的命令 |
| -f filename | 从文件中读取命令 |
| PerfCounter.print | 打印目标Java进程上可用的性能计数器 |

# 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. In source-file mode, 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. In source-file mode, 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. In source-file mode, 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.
      4. See JEP 330: Launch Single-File Source-Code Programs for complete details.

### Standard Options for Java

### Extra Options for Java

### Advanced Options for Java

### Advanced Runtime Options for Java

### Advanced JIT Compiler Options for java

### Advanced Serviceability Options for Java

### Advanced Garbage Collection Options for Java

### Obsolete Java Options

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

### java Command-Line Argument Files

### Example

## 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

### 说明

* + - 1. 反编译二进制文件(class)
      2. 无options时，反编译protected和public字段和方法

### 基础指令

javap [options] classes...

### 主要参数

|  |  |
| --- | --- |
| classes | 1. 指定一个class文件，比如： 2. path/to/MyClass.class 3. jar:file:///path/to/MyJar.jar!/mypkg/MyClass.class 4. java.lang.Object |
| options | |
| - verbose  -v | 打印附加信息 |
| -l | 打印本地变量表 |
| -public | 仅打印public 方法和字段 |
| -protected | 仅打印protected和public方法和字段 |
| -private  -p | 打印所有方法和字段，包括private |
| -c | 反编译代码 |
| -s | 打印内部类 |
| -sysinfo | 展示系统信息 |
| -constants | 显示常量 |
| --module ${module}  -m ${module} | 指定类所在模块 |
| --module-path | 模块路径 |
| --system | 指定系统模块路径 |
| --class-path path  -classpath path  -cp path | 指定类路径，会覆盖CLASSPATH环境变量 |
| -bootclasspath | 指定启动类 |
| --multi-release | 指定jdk版本 |
| -J-${option} | 附带java参数，option参考java指令 |

# 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

### Synopsis

### Example

## jconsole

### Description

### Synopsis

### Example

## jdb

### Description

### Synopsis

### Example

## 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

### Synopsis

### Example

## serialver

### Description

### Synopsis

### Example

# 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)

## 总揽

### 简介

* + - 1. 文档

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，这里基于1.11版本
      2. 基本语句

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

### 概述

* + - 1. 文档

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

* + - 1. 基本语句

**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