

JVM诊断工具

[零 参考资料 4](#_Toc65741365)

[一 JDK分析工具 4](#_Toc65741366)

[1 jps 4](#_Toc65741367)

[1.1 说明 4](#_Toc65741368)

[1.2 文档 4](#_Toc65741369)

[1.3 命令 4](#_Toc65741370)

[1.4 参数说明 4](#_Toc65741371)

[1.5 输出 4](#_Toc65741372)

[2 jstat 4](#_Toc65741373)

[2.1 说明 4](#_Toc65741374)

[2.2 文档 4](#_Toc65741375)

[2.3 命令 4](#_Toc65741376)

[2.4 参数 5](#_Toc65741377)

[2.5 输出 5](#_Toc65741378)

[3 jinfo 7](#_Toc65741379)

[3.1 说明 7](#_Toc65741380)

[3.2 文档 7](#_Toc65741381)

[3.3 命令 7](#_Toc65741382)

[3.4 参数 7](#_Toc65741383)

[4 jmap 8](#_Toc65741384)

[4.1 说明 8](#_Toc65741385)

[4.2 文档 8](#_Toc65741386)

[4.3 命令 8](#_Toc65741387)

[4.4 参数 8](#_Toc65741388)

[5 jhat 8](#_Toc65741389)

[6 jstack 8](#_Toc65741390)

[6.1 说明 8](#_Toc65741391)

[6.2 文档 8](#_Toc65741392)

[6.3 命令 8](#_Toc65741393)

[6.4 参数 8](#_Toc65741394)

[7 jstatd 9](#_Toc65741395)

[7.1 说明 9](#_Toc65741396)

[7.2 文档 9](#_Toc65741397)

[7.3 命令 9](#_Toc65741398)

[7.4 参数 9](#_Toc65741399)

[8 jcmd 9](#_Toc65741400)

[8.1 说明 9](#_Toc65741401)

[8.2 文档 9](#_Toc65741402)

[8.3 命令 9](#_Toc65741403)

[8.4 参数 9](#_Toc65741404)

[二 JVM常用执行指令 9](#_Toc65741405)

[1 java 9](#_Toc65741406)

[1.1 说明 9](#_Toc65741407)

[1.2 文档 9](#_Toc65741408)

[1.3 基础指令 9](#_Toc65741409)

[1.4 参数 10](#_Toc65741410)

[2 javac 17](#_Toc65741411)

[2.1 文档 17](#_Toc65741412)

[2.2 基础指令 17](#_Toc65741413)

[2.3 主要参数 17](#_Toc65741414)

[3 jar 19](#_Toc65741415)

[3.1 说明 19](#_Toc65741416)

[3.2 文档 19](#_Toc65741417)

[3.3 基础指令 19](#_Toc65741418)

[3.4 主要参数 19](#_Toc65741419)

[4 javap 20](#_Toc65741420)

[4.1 说明 20](#_Toc65741421)

[4.2 文档 20](#_Toc65741422)

[4.3 基础指令 20](#_Toc65741423)

[4.4 主要参数 20](#_Toc65741424)

[三 其他命令 21](#_Toc65741425)

[1 jaotc 21](#_Toc65741426)

[2 jarsigner 21](#_Toc65741427)

[3 javadoc 21](#_Toc65741428)

[4 jconsole 21](#_Toc65741429)

[5 jdb 21](#_Toc65741430)

[6 jdeprscan 21](#_Toc65741431)

[7 jdeps 21](#_Toc65741432)

[8 jfr 21](#_Toc65741433)

[9 jhsdb 21](#_Toc65741434)

[10 jlink 21](#_Toc65741435)

[11 jmod 21](#_Toc65741436)

[12 jpackage 21](#_Toc65741437)

[13 jrunscript 21](#_Toc65741438)

[14 jshell 21](#_Toc65741439)

[15 keytool 21](#_Toc65741440)

[16 rmid 21](#_Toc65741441)

[17 rmiregistry 21](#_Toc65741442)

[18 serialver 21](#_Toc65741443)

[四 OQL(对象查询语言) 21](#_Toc65741444)

[1 文档 21](#_Toc65741445)

[2 概述 21](#_Toc65741446)

[3 BNF 22](#_Toc65741447)

[五 Eclipse MAT OQL(1.11.0) 22](#_Toc65741448)

[1 总揽 22](#_Toc65741449)

[1.1 简介 22](#_Toc65741450)

[1.2 SELECT 22](#_Toc65741451)

[1.3 FROM 22](#_Toc65741452)

[1.4 WHERE 23](#_Toc65741453)

[1.5 UNION 24](#_Toc65741454)

[2 Property Accessors 24](#_Toc65741455)

[2.1 Accessing fields of the heap object 24](#_Toc65741456)

[2.2 Calling Java Bean properties： 24](#_Toc65741457)

[2.3 Calling Java methods 25](#_Toc65741458)

[2.4 Array Access 26](#_Toc65741459)

[2.5 Built-in OQL function 27](#_Toc65741460)

[3 Simulated in OQL 27](#_Toc65741461)

[4 More Example 32](#_Toc65741462)

[5 BNF for the Object Query Language 33](#_Toc65741463)

[六 JVisualMV OQL 33](#_Toc65741464)

[1.1 概述 33](#_Toc65741465)

[2 Built-in Object——heap 33](#_Toc65741466)

[2.1 Introduction 33](#_Toc65741467)

[2.2 Example 34](#_Toc65741468)

[3 Functions on individual objects 34](#_Toc65741469)

[3.1 Introduction 34](#_Toc65741470)

[3.2 Example 35](#_Toc65741471)

[4 array/iterator/enumeration manipulation functions 35](#_Toc65741472)

[4.1 Introduction 35](#_Toc65741473)

[4.2 Example 36](#_Toc65741474)

[5 More examples 38](#_Toc65741475)

[七 可视化工具列表 40](#_Toc65741476)

[1 Visual VM 40](#_Toc65741477)

[2 JConsole 40](#_Toc65741478)

[3 Mission Control 40](#_Toc65741479)

[4 JHSDB 40](#_Toc65741480)

[5 Eclipse MAT 40](#_Toc65741481)

# 零 参考资料

《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分析工具

## jps

### Description

类似Linux的ps命令，只列出当前用户的有权限查看的java程序的进程

### Command

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

### Option

|  |  |
| --- | --- |
| -q | 只输出进程ID |
| mlvV | |
| -m | 列出传递给主函数的参数 |
| -l | 主函数完整路径 |
| -v | 列出传递给jvm的参数 |
| -V | 列出通过flags文件传递给jvm的参数 |

### 输出

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

例子：

> jps

* + - 1. 18027 Java2Demo.JAR
      2. 18032 jps

18005 jstat

> jps -l remote.domain

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

2857 sun.tools.jstatd.jstatd

> jps -m remote.domain:2002

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

3102 sun.tools.jstatd.jstatd -p 2002

## jstat

### 说明

查看java运行时信息

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jstat.html

### 命令

jstat [Options] ${pid} [ ${interval}[s|ms] [ ${count} ] ]

### 参数

|  |  |
| --- | --- |
| -help | 帮助信息 |
| -class | 显示classLoader信息 |
| -compiler | 显示jit(即时编译器)相关信息 |
| -gc | 监视gc信息 |
| -gccapacity | 显示各个区域信息 |
| -gccause | 跟gcutil一样，显示垃圾回收信息，但额外显示上一次GC原因 |
| -gcnew | 新生代信息 |
| -gcnewcapacity | 新生代大小 |
| -gcold | 老年代信息和元数据区信息 |
| -gcoldcapacity | 老年代大小 |
| -gcmetacapacity | 显示元数据区大小 |
| -gcutil | 与-gc相同，显示垃圾回收信息，着重于已使用空间的百分比 |
| -printcompilation | 显示jit编译的方法信息 |
| -t | 显示程序运行时间 |
| -h | 指定周期性输出数据， |
| pid | 本地/远程的进程ID  [protocol:][//]***lvmid***[@hostname[:port][/servername]  例如  远程：jstat -gcutil 40496@remote.domain 1000  本地 jstat -gcutil 123 |
| interval | 周期时间，单位为s/ms，默认ms |
| count | 循环次数 |

### 输出

|  |  |
| --- | --- |
| 样例 | 输出说明 |
| jstat -class 1 | Loaded: 加载class数量.  Bytes: 加载class总大小，单位KB.  Unloaded: 被卸载的class数量  Bytes:被卸载的class大小  Time:对class加载&卸载所需要的时间. |
| jstat -compiler 1 | Java HotSpot VM Just-in-Time compiler statistics.  Compiled: 编译任务数量.  Failed: 编译失败任务数  Invalid: 废弃编译任务数  Time: 编译任务执行时间  FailedType: 最后失败的编译任务类型  FailedMethod:最后编译失败的类名和方法 |
| jstat -gc 1 | Garbage collected heap statistics.  S0C: 当前survivor 0容量 (KB).  S1C: 当前survivor 1容量 (KB).  S0U: Survivor 0 使用量(KB).  S1U: Survivor 1 使用量(KB).  EC: 当前eden 容量 (KB).  EU: Eden 使用量 (KB).  OC: old space 容量(KB).  OU: Old space 使用量(KB).  MC: Metaspace 容量 (KB).  MU: Metaspace 使用量(KB).  CCSC: 压缩class 空间容量 (KB).  CCSU: 压缩class 使用量 (KB).  YGC:年轻GC次数.  YGCT: 年轻GC时间.  FGC: FullGC 次数.  FGCT: FullGC时间.  GCT: GC总时间. |
| jstat -gccapacity 1 | Memory pool generation and space capacities.  NGCMN: 最小新时代容量 (KB).  NGCMX: 最大新生代容量 (KB).  NGC: 当前新生代容量 (KB).  S0C: 当前survivor 0 容量(KB).  S1C: 当前survivor 1 容量(KB).  EC: 当前eden 容量(KB).  OGCMN: 最小老年代容量 (KB).  OGCMX: 最大老年代容量(KB).  OGC: 当前老年代容量 (KB).  OC: 当前老年空间容量 (KB).  MCMN: 最小元数据空间 (KB).  MCMX: 最大元数据空间(KB).  MC: 元数据Committed大小 (KB).  CCSMN: 被压缩的类空间最小容量 (KB).  CCSMX: 被压缩的类空间最大容量 (KB).  CCSC: 被压缩的类空间Committed容量(KB).  YGC: 年轻代GC次数  FGC: FullGC次数 |
| jstat -gccause 1 | 显示内容跟-gcutil相同，但多了一下两个信息:  LGCC: 最近一次的GC原因  GCC: 当前GC的原因 |
| jstat -gcnew 1 | 新生代信息统计.  S0C: 当前survivor 0 容量(KB).  S1C: 当前survivor 1 容量(KB).  S0U: 当前survivor 0 使用量(KB).  S1U: 当前survivor 1 使用 (KB).  TT: 晋升老年代阈值  MTT: 晋升老年代最大阈值  DSS: 期望survivor 使用量(KB).  EC: 当前eden space 容量(KB).  EU: Eden space 使用量(KB).  YGC: 年轻代GC次数  YGCT: 年轻代GC时间 |
| jstat -gcnewcapacity 1 | 新生代空间大小统计  NGCMN: 最小新生代容量 (KB).  NGCMX: 最大新生代容量(KB).  NGC: 当前新生代容量(KB).  S0CMX: 最大survivor 0 容量(KB).  S0C: 当前survivor 0 容量(KB).  S1CMX: 最大survivor 1 容量(KB).  S1C: 最大survivor 1 容量(KB).  ECMX: 最大eden 容量 (KB).  EC: 当前eden 容量 (KB).  YGC: 年轻代GC次数.  FGC: FullGC次数 |
| jstat -gcold 1 | 老年代容量统计.  MC: Metaspace Committed Size (KB).  MU: Metaspace 使用量(KB).  CCSC: 压缩类 committed size (KB).  CCSU: 压缩类空间使用量 (KB).  OC: 当前老年空间容量 (KB).  OU: 老年空间使用量 (KB).  YGC: 年轻代GC次数.  FGC: FullGC次数.  FGCT: FullGC时间.  GCT: 所有GC时间. |
| jstat -gcoldcapacity 1 | 老年代统计.  OGCMN: 最小老年代容量 (KB).  OGCMX: 最大老年代容量 (KB).  OGC: 当前老年代容量 (KB).  OC: 当前老年代空间容量 (KB).  YGC: 年轻代GC次数.  FGC: FullGC次数.  FGCT: FullGC时间.  GCT: 所有GC时间. |
| jstat -gcmetacapacity 1 | 元数据空间大小统计  MCMN: 最小元数据空间容量 (KB).  MCMX: 最大元数据空间容量(KB).  MC: 元数据空间Committed 大小(KB).  CCSMN: 压缩类空间最小容量 (KB).  CCSMX: 压缩类空间最大容 (KB).  YGC: 年轻代GC次数.  FGC: FullGC次数.  FGCT: FullGC时间.  GCT: 所有GC时间. |
| jstat -gcutil 1 | GC简要统计  S0: Survivor 0 使用率.  S1: Survivor 1 使用率.  E: Eden space 使用率.  O: 老年空间使用率.  M: 元数据空间使用率.  CCS: 压缩类空间使用率.  YGC: 年轻代GC次数.  YGCT: 年轻代GC时间.  FGC: FULL GC 次数.  FGCT: FULL GC时间.  GCT: 所有GC时间. |
| jstat -printcompilation 1 | Java HotSpot VM 编译统计.  Compiled:最近一次执行编译方法的任务数量.  Size: 最近一次编译方法的字节码总字节数（byte）.  Type: 最近一次编译方法的变异类型.  Method: 最近一次编译的方法的唯一限定名 |

## jinfo

### 说明

查看java的参数，支持在运行时修改部分参数，这个命令处于实验性质

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jinfo.html

### 命令

jinfo [option] pid

### 参数

|  |  |
| --- | --- |
| no-option | 输出全部的参数和系统属性 |
| -flag<name> | 打印指定参数 |
| -flag [+|-]<name> | 开启或关闭指定参数, Boolean |
| -flag <name>=<value> | 设置指定jvm参数值 |
| - sysprops | 输出系统属性 |

## jmap

### 说明

可以生产Java程序的堆dump文件，也可以查看堆统计信息、classLoader信息、finalizer队列

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jmap.html

### 命令

jmap [options] pid

### 参数

|  |  |
| --- | --- |
| no option |  |
| -histo[:live] | 打印每个class的实例数目,内存占用,类全名信息. VM的内部类名字开头会加上前缀”\*”. 如果live子参数加上后,只统计活的对象数量 |
| -dump:***dump\_options*** | 使用hprof二进制形式,输出jvm的heap内容到文件  ***dump\_options***取值：  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  例子 |
| -heap | 打印堆摘要、配置、gc算法，打印字符串的数量和大小。 |
| -clstats | 打印classLoader信息 |
| -F | 在没有pid时，强制使用-histo和-dump，同时不支持live |
| -h | 帮助信息 |
| -help | 帮助信息 |
| -Jflag | 传递参数给jmap启动的jvm |

## 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

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jstack.html

### 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

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jstatd.html

### 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

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jcmd.html

### 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 |

# JVM常用执行指令

## java

### 说明

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/java.html

### 基础指令

* + - 1. 执行一个main类

java [options] mainclass [args ...]

* + - 1. 执行一个带main方法的jar包

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

* + - 1. 执行一个模块里的main类

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

java [options] --module module[/mainclass] [args ...]

* + - 1. 执行一个源文件，即非class二进制文件，如：hello.java

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

ps:这个语句等同于，javac hello.java & java hello

### 参数

ps：如无特殊说明，非windows指的是**Oracle Solaris, Linux, and macOS**平台

|  |  |
| --- | --- |
| classname | 运行的class文件 |
| args | main方法入参 |
| options | |
| -version[:release] |  |
| -jar | 执行一个jar包 |
| -m  --module | 执行一个模块，jdk9开始有模块概念 |
| **标准 Options，适用于所有虚拟机实现** | |
| -agentlib:libname[=options] | 加载本地代理库 |
| -agentpath:pathname[=options] | 加载本地代理库，机制跟agentlib相同，但需要库的绝对路径 |
| --class-path classpath  -classpath  -cp | 指定class库，用（;）分割多个  等同环境变量(CLASSPATH) |
| --disable-@files | 禁用某个文件 |
| --enable-preview | 是否可以用预览功能 |
| --module-path modulepath...  -p modulepath | 模块路径，用（;）分割多个 |
| --upgrade-module-path modulepath... | 替换指定模块 |
| --add-modules ${module} | 指定根模块，module可以是ALL-DEFAULT, ALL-SYSTEM, ALL-MODULE-PATH. |
| --list-modules | 仅列出可见模块并退出 |
| -d module\_name  --describe-module module\_name | 仅显示指定模块的描述并退出 |
| --dry-run | 仅启动VM但不运行main方法，用于测试option参数是否正确 |
| --validate-modules | 仅验证模块冲突和错误，并退出 |
| -Dproperty=value | 设置一个系统属性值，值如果有空格需要双引号包裹 |
| -disableassertions[:[packagename]...|:classname]  -da[:[packagename]...|:classname] | 禁用断言，默认范围为所有包和类  可以通过指定包和类。如果参数是:...，则针对无名包禁用断言 |
| -disablesystemassertions  -dsa | 禁用所有系统类的断言 |
| -enableassertions[:[packagename]...|:classname]  -ea[:[packagename]...|:classname] | 启用断言，默认范围为所有包和类  可以通过指定包和类。如果参数是:...，则针对无名包启用断言 |
| -enablesystemassertions  -esa | 启用所有系统类的断言 |
| -javaagent:jarpath[=options] | 指定代理包 |
| --show-version | 启动时打印版本号 |
| -showversion | 抛错时打印版本号 |
| --show-module-resolution | 启动期间显示模块间关系 |
| -splash:imagepath | 指定一个splash图片 |
| -verbose:class | 显示加载的类的信息 |
| -verbose:gc | 显示GC事件信息 |
| -verbose:jni | 显示调用native方法或接口行为的信息 |
| -verbose:module | 显示使用到的模块信息 |
| --version | 打印版本信息并退出 |
| -version | 打印版本信息并退出 |
| -X | 打印扩展option的帮助信息 |
| --help-extra | 打印扩展option的帮助信息 |
| @argfile | 从文件中读取option配置 |
| **扩展Option，仅适用于HotSpot** | |
| -Xbatch | 禁用后台编译  等同于-XX:-BackgroundCompilation |
| -Xbootclasspath/a:directories|zip|JAR-files | 指定引导类路径，多个时：  windows用;分割  非windows用:分割 |
| -Xcheck:jni | 在调用jni之前会检查调用参数和运行时环境数据，  也会检查挂起异常，  会降低性能 |
| -Xdebug | 什么都不做，向后兼容 |
| -Xdiag | 显示附加的诊断信息 |
| -Xint | 仅用解释模式执行  禁止编译native代码  解释执行所有字节  JIT编译的优势无法体现 |
| -Xinternalversion | 显示比-version更多的信息 |
| -Xlog:option | 启用统一日志组件  详见[链接](https://docs.oracle.com/en/java/javase/14/docs/specs/man/java.html#enable-logging-with-the-jvm-unified-logging-framework) |
| -Xmixed | 用解释模式执行字节码，不影响native code |
| -Xmn size | 设置年轻代初始和最大堆容量，  单位：k，m，g  可以分别用-XX:NewSize设置初始大小，-XX:MaxNewSize设置最大大小 |
| -Xms size | 设置最小&初始堆大小  值必须1024的整数倍，大于1MB  可以分别用-XX:MinHeapSize设置最小堆大小，-XX:InitialHeapSize设置堆初始大小  如果不设置此值，默认初始化堆大小为年轻代和年老代的和 |
| -Xmx size | 指定堆的最大值  值必须1024的整数倍，大于2MB  默认值决定于运行时系统配置  等同于-XX:MaxHeapSize. |
| -Xnoclassgc | 禁用gc |
| -Xrs | 减少JVM感知系统信号  solaris,linux,macos中，减少SIGINT, SIGTERM, SIGHUP, and SIGQUIT信号干扰，同时无法导出SIGQUIT 线程信息  windows中减少CTRL\_C\_EVENT, CTRL\_CLOSE\_EVENT, CTRL\_LOGOFF\_EVENT, or CTRL\_SHUTDOWN\_EVENT干扰，同时无法用ctrl+break导出线程 |
| -Xshare:mode | 设置class data sharing(CDS)  mode取值：  auto：默认  on，强制使用CDS，否则抛错；仅用于测试，不能用于生产环境  off：关闭CDS |
| -XshowSettings | 显示所有设置 |
| -XshowSettings:category | 显示指定类型设置  category取值：  all：默认值  locale：本地设置  properties：系统设置  vm：jvm设置  system：适用于linux，显示主机系统或容器配置 |
| -Xss size | 设置线程本地栈大小  windows默认值以来虚拟内存  非windows：1024KB  等同于-XX:ThreadStackSize |
| --add-reads module=target-module | 无条件让指定模块读取目标模块(Updates module to read the target-module, regardless of the module declaration) |
| --add-exports module/package=target-module | 无条件将模块导出包到目标模块的 |
| --add-opens module/package=target-module | 无条件将模块的包开放给目标模块 |
| --illegal-access=parameter | 非法访问数据：  permit：默认行为。允许对封装类型进行非法访问。当第一次尝试通过反射进行非法访问时会生成一个警告  warn：与permit一样，但每次非法访问尝试时都会产生错误  debug：同时显示非法访问尝试的堆栈跟踪。  deny：不允许非法的访问尝试。这将是未来的默认行为 |
| --limit-modules module[,module...] | 指定可见模块范围 |
| --patch-module module=file(;file)\* | 重写指定模块 |
| --source version | 设置数据源版本 |
| **进阶option** | |
| -XX:+UnlockDiagnosticVMOptions | 启用诊断类配置，默认是禁用 |
| -XX:+UnlockExperimentalVMOptions | 启用实验性质的功能 |
| **HotSpot进阶配置** | |
| -XX:ActiveProcessorCount=x | 修改jvm可用cpu核数，受-XX:-UseContainerSupport配置影响 |
| -XX:AllocateHeapAt=path | 指定对象堆分配位置，  代码、元数据、线程本地栈仍在内存中，不影响 |
| -XX:-CompactStrings | 禁用压缩字符串功能，启用压缩功能时，在内部仅用单字节表示并用ISO-8859-1 / Latin-1编码表示每个字符(When this option is enabled, Java Strings containing only single-byte characters are internally represented and stored as single-byte-per-character Strings using ISO-8859-1 / Latin-1 encoding)。  禁用压缩后，使用utf-16编码(jdk8是utf-8) |
| -XX:ErrorFile=filename | 指定错误数据输出文件  默认是当前目录，名为hs\_err\_pidpid.log文件 |
| -XX:+ExtensiveErrorReports | 输出更详细错误报告到ErrorFile，包括敏感信息，默认禁用 |
| -XX:FlightRecorderOptions=parameter=value  -XX:FlightRecorderOptions:parameter=value | 配置jfr参数，商业工具，具体参数值参考文档 |
| -XX:LargePageSizeInBytes=size | 设置大页容量，对内存密集型程序有利，必须是2的倍数 |
| -XX:MaxDirectMemorySize=size | 配置直接内存大小，默认或设置为0，JVM自动分配 |
| -XX:-MaxFDLimit | 禁用打开文件描述符限制改为硬件限制(Disables the attempt to set the soft limit for the number of open file descriptors to the hard limit)，只有在macos下才需要禁用。 |
| -XX:NativeMemoryTracking=mode | 指定跟踪本地内存模式  mode取值：  off，默认值，不跟踪本地内存使用  summary，只跟踪基本内存使用  detail，跟踪内存使用详情 |
| -XX:ObjectAlignmentInBytes=alignment | 对象大小对齐，取值8-256，2的倍数；默认8byte； |
| -XX:OnError=string | 在抛错时，执行指定命令串，多命令用分号分隔  比如非windows：  -XX:OnError="gcore %p;gdb -p %p" |
| -XX:OnOutOfMemoryError=string | 在OOM时执行指定命令 |
| -XX:+PrintCommandLineFlags | 打印JVM显示和隐式(默认)参数 |
| -XX:+PreserveFramePointer | 使用RBP寄存器存储frame pointer |
| -XX:+PrintNMTStatistics | 当设置-XX:NativeMemoryTracking，在JVM退出时收集打印本地内存跟踪数据 |
| -XX:SharedArchiveFile=path | 指定CDS文件路径 |
| -XX:SharedArchiveConfigFile=shared\_config\_file | 将分享数据加入到指定的配置文件中 |
| -XX:SharedClassListFile=file\_name | 批量加载存储在指定文件里的CDS列表 |
| -XX:+ShowCodeDetailsInExceptionMessages | 显示详细错误信息，for NullPointerException |
| -XX:+ShowMessageBoxOnError | 当虚拟机出错时，弹出一个提示窗，并阻塞退出操作，用于分析出错原因 |
| -XX:StartFlightRecording=parameter=value | 启用jfr，商业工具，具体参数值参考文档 |
| -XX:ThreadStackSize=size | 设置线程栈大小，同-Xss |
| -XX:-UseBiasedLocking | 禁用偏向锁 |
| -XX:-UseCompressedOops | 禁用压缩指针，默认启用压缩指针，仅在64bit的JVM有效 |
| -XX:-UseContainerSupport | 禁用容器支持，仅在64bit的JVM有效 |
| -XX:+UseHugeTLBFS | 仅用于LINUX，等同于  -XX:+UseLargePage |
| -XX:+UseLargePages | 启用大型页 |
| -XX:+UseTransparentHugePages | 仅用于Linux，启用动态分配large pages |
| -XX:+AllowUserSignalHandlers | 允许用户处理应用消息 |
| -XX:VMOptionsFile=filename | 指定VM配置文件 |
| **基于HotSpot的JIT相关高级参数** | |
| -XX:AllocateInstancePrefetchLines=lines | 设置实例分配前预留缓存行数，默认1 |
| -XX:AllocatePrefetchDistance=size | 设置对象预分配空间，默认值-1 |
| -XX:AllocatePrefetchInstr=instruction | 在分配指针前预取指令数，默认0 |
| -XX:AllocatePrefetchLines=lines | 设置分配对象后预留缓存行，如果最后个对象非数组，默认1，否则是3 |
| -XX:AllocatePrefetchStepSize=size | 设置顺序指令每步骤预留空间大小，默认16b |
| -XX:AllocatePrefetchStyle=style | 产生预取指令方式  0:不发生预取指令  1:默认配置，在每次分配后，执行预取指令  2:用TLAB watermark pointer决定预取指令  3：每个缓存行产生一条预取指令 |
| -XX:+BackgroundCompilation | 启用后台编译 |
| -XX:CICompilerCount=threads | 设置编译线程数量  默认值是CPU数和内存大小 |
| -XX:+UseDynamicNumberOfCompilerThreads | 动态创建编译线程 |
| -XX:CompileCommand=command,method[,option] | 在执行指定方法时，执行指定命令 |
| -XX:CompileCommandFile=filename | 批量设置在执行指定方法时，执行指定命令 |
| -XX:CompilerDirectivesFile=file | 在程序启动是将指定指令加载到指令栈  需要配合  -XX:UnlockDiagnosticVMOptions |
| -XX:+CompilerDirectivesPrint | 当项目启动或指令加载时，打印指令栈 |
| -XX:CompileOnly=methods | 只编译指定的方法，多个用都好隔开 |
| -XX:CompileThresholdScaling=scale | 指定方法首次编译方式 |
| -XX:+DoEscapeAnalysis | 启用逃逸分析 |
| -XX:InitialCodeCacheSize=size | 设置初始缓存大小 |
| -XX:+Inline | 启用方法内联 |
| -XX:InlineSmallCode=size | 设置内联代码量 |
| -XX:+LogCompilation | 记录编译行为日志 |
| -XX:FreqInlineSize=size | 设置hot方法内联空间 |
| -XX:MaxInlineSize=size | 设置cold方法内联空间 |
| -XX:MaxTrivialSize=size | 设置trivial方法内联空间 |
| -XX:MaxNodeLimit=nodes | 设置单方法编译时节点数 |
| -XX:NonNMethodCodeHeapSize=size | 设置代码段包含非代码容量 |
| -XX:NonProfiledCodeHeapSize=size | 设置代码段包含无关方法大小  仅在设置-XX:SegmentedCodeCache时生效 |
| -XX:+OptimizeStringConcat | 优化string拼接 |
| -XX:+PrintAssembly | 打印汇编码 |
| -XX:ProfiledCodeHeapSize=size | 设置代码段包含profiled methods大小，  仅在设置-XX:SegmentedCodeCache时生效 |
| -XX:+PrintCompilation | 打印所有方法编译时日志 |
| -XX:+PrintInlining | 打印内联日志，可以查看哪些方法内联处理 |
| -XX:ReservedCodeCacheSize=size | 预留代码缓存，不能比  -XX:InitialCodeCacheSize小 |
| -XX:RTMAbortRatio=abort\_ratio | 指定RTM(Restricted Transactional Memory)中止比例 |
| -XX:RTMRetryCount=number\_of\_retries | 指定RTM 锁定代码重试次数 |
| -XX:+SegmentedCodeCache | 允许代码段缓存 |
| -XX:StartAggressiveSweepingAt=percent | 指定执行强制移除未使用代码的剩余空间阈值比例 |
| -XX:-TieredCompilation | 禁用分层编译 |
| -XX:UseSSE=version | 指定sse指令版本，仅适用86 |
| -XX:UseAVX=version | 指定AVX指令仅适用86 |
| -XX:+UseAES | 使用基于硬件的编译器内部AES |
| -XX:+UseAESIntrinsics | 使用编译器内部AES |
| -XX:+UseAESCTRIntrinsics | 使用编译器内部AES/CTR |
| -XX:+UseGHASHIntrinsics | 使用编译器内部GHASH，需要同时设置  -XX:+UnlockDiagnosticVMOptions |
| -XX:+UseBASE64Intrinsics | 使用编译器内部BASE64  -XX:+UnlockDiagnosticVMOptions |
| -XX:+UseAdler32Intrinsics | 使用编译器内部Adler32  -XX:+UnlockDiagnosticVMOptions |
| -XX:+UseCRC32Intrinsics | 同上 |
| -XX:+UseCRC32CIntrinsics | 同上 |
| -XX:+UseSHA | 同上 |
| -XX:+UseSHA1Intrinsics | 同上 |
| -XX:+UseSHA256Intrinsics | 同上 |
| -XX:+UseSHA512Intrinsics | 同上 |
| -XX:+UseMathExactIntrinsics | 同上 |
| -XX:+UseMultiplyToLenIntrinsic | 同上 |
| -XX:+UseSquareToLenIntrinsic | 同上 |
| -XX:+UseMulAddIntrinsic | 同上 |
| -XX:+UseMontgomeryMultiplyIntrinsic | 同上 |
| -XX:+UseMontgomerySquareIntrinsic | 同上 |
| -XX:+UseCMoveUnconditionally | 无条件生成CMove指令 |
| -XX:+UseCodeCacheFlushing | 关闭编译时，清空代码缓存 |
| -XX:+UseCondCardMark | 启用更新卡表前检查卡是否被标志，默认是关闭。在并发操作时，能提高性能 |
| -XX:+UseCountedLoopSafepoints | 将安全点保存在统计循环(counted loop)中 |
| -XX:LoopStripMiningIter=number\_of\_iterations | Controls the number of iterations in the inner strip mined loop |
| -XX:LoopStripMiningIterShortLoop=number\_of\_iterations | Controls loop strip mining optimization |
| -XX:+UseFMA | 使用编译器内部基于硬件的FMA指令 |
| -XX:+UseRTMDeopt | 自动调节RTM锁定 |
| -XX:+UseRTMLocking | 用RTM为重量级锁锁定代码 |
| -XX:+UseSuperWord | 向量化scalar操作 |
| **可用性相关参数** | |
| -XX:+DisableAttachMechanism | 禁用分析工具，如jcmd, jstack, jmap, jinfo等 |
| -XX:+ExtendedDTraceProbes | 启用动态追踪，仅非Win系统 |
| -XX:+HeapDumpOnOutOfMemoryError | 在OOM时，导出堆信息 |
| -XX:HeapDumpPath=path | 指定堆导出文件存储路径，默认为当前工作目录，文件名默认为java\_pid<pid>.hprof |
| -XX:LogFile=path | 设置虚拟机日志路径，默认为当前工作目录，默认文件名为hotspot.log. |
| -XX:+PrintClassHistogram | 打印类直方图，当收到指定系统事件时：  win：control+c  非win：control+break |
| -XX:+PrintConcurrentLocks | 打印锁信息，当收到同上系统事件时 |
| -XX:+PrintFlagsRanges | 打印测试信息 |
| -XX:+PerfDataSaveToFile | 当应用退出时，保存状态二进制信息到指定文件 |
| -XX:+UsePerfData | 启用perfdata功能 |
| **高级HotSpot参数** | |
| -XX:+AggressiveHeap | 根据CPU和RAM自动优化heap |
| -XX:+AlwaysPreTouch | 在OS请求之后，分配内存给应用之前，VM移动每一页(Requests the VM to touch every page on the Java heap after requesting it from the operating system and before handing memory out to the application.) |
| -XX:ConcGCThreads=threads | 并发GC线程数，默认是 |
| -XX:+DisableExplicitGC | 禁用System.gc() |
| -XX:+ExplicitGCInvokesConcurrent | System.gc()立即触发GC，仅在设置-XX:+UseG1GC时有效 |
| -XX:G1AdaptiveIHOPNumInitialSamples=number | 设置采集标本周期数， |
| -XX:G1HeapRegionSize=size | 设置G1堆的域大小，必须是2的倍数，区间1M-32M |
| -XX:G1HeapWastePercent=percent | 浪费堆内存比例，即当可回收比例小于此值时，不执行混合GC周期，默认5 |
| -XX:G1MaxNewSizePercent=percent | G1年轻代比例，默认60 |
| -XX:G1MixedGCCountTarget=number | 混合GCtarget数，默认8 |
| -XX:G1MixedGCLiveThresholdPercent=percent | 混合GC时，老年代占用比例，默认85 |
| -XX:G1NewSizePercent=percent | 新生代最小比例，默认5 |
| -XX:G1OldCSetRegionThresholdPercent=percent | 每个混合GC周期收集老年代比例，默认10 |
| -XX:G1ReservePercent=percent | 预留堆比例，默认10 |
| -XX:+G1UseAdaptiveIHOP | 使用自适应老年代占用空间 |
| -XX:InitialHeapSize=size | 堆初始化，值要么0，要么1024的倍数并大于1MB。当赋值0时，大小为新老两代空间之和 |
| -XX:InitialRAMPercentage=percent | 初始堆占用内存比例，默认值为1.5625 |
| -XX:InitialSurvivorRatio=ratio | Survivor区初始比例，默认值8，在设置-XX:+UseParallelGC 或-XX:+UseParallelOldGC时有效 |
| -XX:InitiatingHeapOccupancyPercent=percent | G1收集器老年代初始占用比例，默认45 |
| -XX:MaxGCPauseMillis=time | G1最大暂停时间，单位毫秒，默认值200 |
| -XX:MaxHeapSize=size | 最大堆，必须是1024的倍数且大于2M |
| -XX:MaxHeapFreeRatio=percent | 允许堆空闲比例，如果堆空闲大于此比例，则缩小堆，默认值是70 |
| -XX:MaxMetaspaceSize=size | 最大元数据空间，默认情况下无上限 |
| -XX:MaxNewSize=size | 新生代最大值 |
| -XX:MaxRAM=size | 最大物理内存占用空间，默认值是最大物理内存和128GB取最小值 |
| -XX:MaxRAMPercentage=percent | 最大物理内存占用 |
| -XX:MinRAMPercentage=percent | 最小物理内存占用 |
| -XX:MaxTenuringThreshold=threshold | 年轻代到老年代的阈值，最大/默认都是15 |
| -XX:MetaspaceSize=size | 设置元数据空间大小 |
| -XX:MinHeapFreeRatio=percent | 设置堆最小空闲比例，低于此比例，堆会扩大 |
| -XX:MinHeapSize=size | 设置堆最小值，值必须是0或者1024倍数且大于1MB，如果值为0，值与初始大小一样 |
| -XX:NewRatio=ratio | 新老年代比例，默认值为2 |
| -XX:NewSize=size | 年轻代初始大小，等同于-Xmn |
| -XX:ParallelGCThreads=threads | STW工作线程并发数 |
| -XX:+ParallelRefProcEnabled | 启用并发引用处理，默认禁用 |
| -XX:+PrintAdaptiveSizePolicy | 启用打印自适应大小信息，默认禁用 |
| -XX:+ScavengeBeforeFullGC | 启用每次FullGC前先尝试回收年轻代 |
| -XX:SoftRefLRUPolicyMSPerMB=time | 设置软引用在最后次被引用后存活时间，单位毫秒， |
| -XX:-ShrinkHeapInSteps | 禁用逐步缩小堆空间，每次都立即缩小到指定大小 |
| -XX:StringDeduplicationAgeThreshold=threshold | 设置String对象去重生效年龄，默认值是3 |
| -XX:SurvivorRatio=ratio | eden和survivor空间比例，默认是8 |
| -XX:TargetSurvivorRatio=percent | 设置年轻代GC后survivor空间比例，默认是50 |
| -XX:TLABSize=size | TLAB大小 |
| -XX:+UseAdaptiveSizePolicy | 开启自适应策略(Ergonomics) |
| -XX:+UseG1GC | 启用G1GC |
| -XX:+UseGCOverheadLimit | 启用GC消耗时间限制，如果GC时间消耗占用了98%且少于2%堆释放，则OOM |
| -XX:+UseNUMA | 使用NUMA（nonuniform memory architecture），默认关闭，仅在parallelGC下有效 |
| -XX:+UseParallelGC | 使用parallel scavenge GC |
| -XX:+UseSerialGC | 使用Serial GC |
| -XX:+UseSHM | 允许JVM使用共享存储设置大页，仅用于Linux |
| -XX:+UseStringDeduplication | 启用String对象去重，仅限G1有效 |
| -XX:+UseTLAB | 启用TLAB |
| -XX:+UseZGC | 启用ZGC |

## javac

### 文档

https://docs.oracle.com/en/java/javase/14/docs/specs/man/javac.html

### 基础指令

javac [options] [sourcefiles-or-classnames]

### 主要参数

|  |  |
| --- | --- |
| **Option** | |
| @filename | 从指定文件加载参数 |
| -A*key*[=value] | 配置注释处理器的参数，不被javac直接解释，如果有多个key/val，用点(.)分割 |
| --add-modules module | 指定附加模块，多个用逗号隔开 |
| --boot-class-path path  -bootclasspath path | 指定重写的引导类，仅适用与jdk9之前 |
| --class-path path  -classpath path  -cp path | 指定类库，会覆盖环境变量CLASSPATH |
| -d directory | 指定class文件存放根目录，如果不指定，则会存到源码文件所在目录 |
| -deprecation | 显示每个deprecated成员和类 |
| --enable-preview | 启用预览功能 |
| -encoding encoding | 指定编码 |
| -endorseddirs directories | 仅适用于jdk9之前 |
| -extdirs directories | 扩展包目录  多个目录：win用分号；分割，非win用冒号分割 |
| -g | 生成所有debug信息，默认情况下仅生成源文件名和行数 |
| -g: [lines, vars, source,none] | 生成指定的debug信息  lines：行号  vars：本地变量  source：源码文件  none:不输出 |
| -h directory | 指定头文件存放目录 |
| --help  -help  -? | 显示标准参数 |
| --help-extra  -X | 显示扩展参数 |
| -implicit:[none , class] | 是否生成隐式引用的类文件  none：不生成  class：自动生成 |
| -J*option* | 传送java命令的参数 |
| --limit-modules module | 限制可见模块，多个用逗号隔开 |
| --module module-name  -m module-name | 覆盖编译指定模块源文件 |
| --module-path path or -p path | 指定模块库路径 |
| --module-source-path module-source-path | 指定模块源码文件 |
| --module-version version | 指定模块版本 |
| -nowarn | 禁止警告信息 |
| -parameters | 生成方法参数元数据，用于可以通过反射获取方法参数原名 |
| -proc:[none,only] | 控制是否完成注解和编译  none：编译时不处理注解  only：只完成注解处理 |
| -processor class1 | 指定注解处理器，多个用逗号分开 |
| --processor-module-path path | 指定注解处理器所在模块 |
| --processor-path path  -processorpath path | 指定注解处理器所在目录 |
| -profile profile | 指定配置文件，仅适用于jdk9之前 |
| --release release | 指定jdk版本，此参数与—source和target参数互斥，同时--add-exports指定的资源对应的jdk版本不能高于此参数值 |
| -s directory | 指定生成的源文件存放位置 |
| --source release  -source release | 指定jdk版本 |
| --source-path path  -sourcepath path | 指定源码文件位置 |
| --system jdk | none | 重写系统模块位置 |
| --target release  -target release | 生成class文件兼容指定jdk版本，此参数值必须大于等于—source参数的值 |
| --upgrade-module-path path | 指定最新包 |
| -verbose | 打印编译冗余编译信息 |
| --version  -version | 打印版本信息 |
| -Werror | 发生错误时，中止编译 |
| --add-exports module/package=other-modules | 指定一个包视为从module导出到other-module，当other-module值为ALL-UNNAMED时，视为导出到所有匿名模块 |
| --add-reads module=other-modules | 将附加的包视为指定包 |
| --default-module-for-created-files module-name | 指定默认注解处理器的模块 |
| -Djava.endorsed.dirs=dirs | 指定重写的标准类包所在目录，仅适用于jdk9之前 |
| -Djava.ext.dirs=dirs | 指定扩展jar包目录，仅适用于jdk9之前 |
| --doclint-format [html4|html5] | 指定注释输出文档格式 |
| --patch-module module=path | 覆盖或增加模块 |
| -Xbootclasspath:path | 指定引导类路径，仅适用于jdk9之前 |
| -Xbootclasspath/a:path | 指定引导类路径后缀，仅适用于jdk9之前 |
| -Xbootclasspath/p:path | 指定引导类路径前缀，仅适用于jdk9之前 |
| -Xdiags:[compact，verbose] | 指定诊断模式，  compact：简略  verbose：详细 |
| -Xdoclint | 启用检查javadoc内容 |
| -Xdoclint:(all|none|[-]group)[/access] | 启用检查指定组的javadoc内容  group可取值：accessibility/html/missing/reference/syntax  access可取值：public/protected/package/private |
| -Xlint | 启用警告信息 |
| -Xlint:[-]key(,[-]key) | 启用指定警告信息，子参数太多了，参考[doc](https://docs.oracle.com/en/java/javase/14/docs/specs/man/javac.html#option-Xlint-custom) |
| -Xmaxerrs number | 打印错误信息数量上限 |
| -Xmaxwarns number | 打印警告信息数量上限 |
| -Xpkginfo:[ always, legacy, nonempty] | 指定生成包信息方式  always:编译每个包信息文件  legacy：只编译有注解的包信息  nonempty：只编译有RetentionPolicy.CLASS 或 RetentionPolicy.RUNTIME注解的包信息 |
| -Xplugin:name args | 加载指定插件 |
| -Xprefer:[source , newer] | 指定隐式编译优先读取方式  source：优先读取源码文件  newer：选取最新class或源文件 |
| -Xprint | 打印debug信息 |
| -XprintProcessorInfo | 打印注解处理信息 |
| -XprintRounds | 打印初始信息和注解处理过程信息 |
| -Xstdout filename | 输出编译信息到指定文件 |
| **环境变量** | |
| CLASSPATH |  |
| JDK\_JAVAC\_OPTIONS |  |

## 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