

JVM诊断工具

[零 参考资料 4](#_Toc64980690)

[一 JDK分析工具 4](#_Toc64980691)

[1 jps 4](#_Toc64980692)

[1.1 说明 4](#_Toc64980693)

[1.2 文档 4](#_Toc64980694)

[1.3 命令 4](#_Toc64980695)

[1.4 参数说明 4](#_Toc64980696)

[1.5 输出 4](#_Toc64980697)

[2 jstat 4](#_Toc64980698)

[2.1 说明 4](#_Toc64980699)

[2.2 文档 4](#_Toc64980700)

[2.3 命令 4](#_Toc64980701)

[2.4 参数 5](#_Toc64980702)

[2.5 输出 5](#_Toc64980703)

[3 jinfo 7](#_Toc64980704)

[3.1 说明 7](#_Toc64980705)

[3.2 文档 7](#_Toc64980706)

[3.3 命令 7](#_Toc64980707)

[3.4 参数 7](#_Toc64980708)

[4 jmap 8](#_Toc64980709)

[4.1 说明 8](#_Toc64980710)

[4.2 文档 8](#_Toc64980711)

[4.3 命令 8](#_Toc64980712)

[4.4 参数 8](#_Toc64980713)

[5 jhat 8](#_Toc64980714)

[6 jstack 8](#_Toc64980715)

[6.1 说明 8](#_Toc64980716)

[6.2 文档 8](#_Toc64980717)

[6.3 命令 8](#_Toc64980718)

[6.4 参数 8](#_Toc64980719)

[7 jstatd 9](#_Toc64980720)

[7.1 说明 9](#_Toc64980721)

[7.2 文档 9](#_Toc64980722)

[7.3 命令 9](#_Toc64980723)

[7.4 参数 9](#_Toc64980724)

[8 jcmd 9](#_Toc64980725)

[8.1 说明 9](#_Toc64980726)

[8.2 文档 9](#_Toc64980727)

[8.3 命令 9](#_Toc64980728)

[8.4 参数 9](#_Toc64980729)

[二 OQL(对象查询语言) 9](#_Toc64980730)

[1 简介 9](#_Toc64980731)

[1.1 文档 9](#_Toc64980732)

[1.2 概述 9](#_Toc64980733)

[三 Eclipse MAT OQL(1.11.0) 10](#_Toc64980734)

[1 总揽 10](#_Toc64980735)

[1.1 简介 10](#_Toc64980736)

[1.2 SELECT 10](#_Toc64980737)

[1.3 FROM 10](#_Toc64980738)

[1.4 WHERE 11](#_Toc64980739)

[1.5 UNION 11](#_Toc64980740)

[2 获取属性 11](#_Toc64980741)

[2.1 获取对象在堆中field： 11](#_Toc64980742)

[2.2 获取对象内置JAVA相关属性： 11](#_Toc64980743)

[2.3 调取java 方法 12](#_Toc64980744)

[2.4 读取数组 12](#_Toc64980745)

[2.5 内置OQL方法 13](#_Toc64980746)

[2.6 例子 13](#_Toc64980747)

[3 模拟SQL中一些实现 13](#_Toc64980748)

[3.2 BNF for the Object Query Language 16](#_Toc64980749)

[四 JVisualMV OQL 16](#_Toc64980750)

[1.1 概述 16](#_Toc64980751)

[1.2 select 16](#_Toc64980752)

[1.3 from 16](#_Toc64980753)

[1.4 where 16](#_Toc64980754)

[1.5 读取数组元素 17](#_Toc64980755)

[2 内置对象 17](#_Toc64980756)

[2.1 对象 17](#_Toc64980757)

[3 例子 17](#_Toc64980758)

[4 对象函数 17](#_Toc64980759)

[5 集合操作 18](#_Toc64980760)

[6 内置方法 19](#_Toc64980761)

[7 例子 19](#_Toc64980762)

[五 JVM常用执行指令 20](#_Toc64980763)

[1 java 20](#_Toc64980764)

[1.1 说明 20](#_Toc64980765)

[1.2 文档 20](#_Toc64980766)

[1.3 基础指令 20](#_Toc64980767)

[1.4 参数 21](#_Toc64980768)

[2 javac 28](#_Toc64980769)

[2.1 文档 28](#_Toc64980770)

[2.2 基础指令 28](#_Toc64980771)

[2.3 主要参数 28](#_Toc64980772)

[3 jar 30](#_Toc64980773)

[3.1 说明 30](#_Toc64980774)

[3.2 文档 30](#_Toc64980775)

[3.3 基础指令 30](#_Toc64980776)

[3.4 主要参数 30](#_Toc64980777)

[4 javap 31](#_Toc64980778)

[4.1 说明 31](#_Toc64980779)

[4.2 文档 31](#_Toc64980780)

[4.3 基础指令 31](#_Toc64980781)

[4.4 主要参数 31](#_Toc64980782)

[六 可视化工具列表 32](#_Toc64980783)

[1 Visual VM 32](#_Toc64980784)

[2 JConsole 32](#_Toc64980785)

[3 Mission Control 32](#_Toc64980786)

[4 JHSDB 32](#_Toc64980787)

# 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/14/

因为我的私服是centos，所以Linux命令是基于centos的linux内核

# JDK分析工具

## jps

### 说明

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

### 文档

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

### 命令

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

### 参数说明

|  |  |
| --- | --- |
| -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 |

# OQL(对象查询语言)

## 文档

## 概述

* + - 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. 本文采用eclipse的MAT工具和 JvisualVM
      3. BNF(Backus-Naur Form)，
         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。

## BNF

# 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/queryingheapobjects.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. DISTINCT:去重
      2. OBJECTS：Use the OBJECTS keyword if you want to process the text that follows the FROM keyword as objects instead of classes.
      3. AS RETAINED SET: 查询结果封装成set集合
      4. 例子
         1. SELECT toString(s) AS Value, s.@usedHeapSize AS "Shallow Size", s.@retainedHeapSize AS "Retained Size" FROM java.lang.String s
         2. SELECT AS RETAINED SET \* FROM java.lang.String
         3. SELECT DISTINCT OBJECTS classof(s) FROM java.lang.String s
         4. SELECT DISTINCT \* FROM OBJECTS 0,1,1,2
         5. SELECT OBJECTS dominators(s) FROM java.lang.String s

### FROM

* + - 1. INSTANCEOF: 指定父类
      2. OBJECTS：强制结果输出为object class list
      3. 根据class查询：

SELECT \* FROM java.lang.String

* + - 1. 正则匹配类名

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

* + - 1. 根据对象地址查询，多个地址逗号隔开

SELECT \* FROM 0x2b7468c8

SELECT \* FROM 0x2b7468c8,0x2b74aee0

* + - 1. 子查询

SELECT \* FROM ( SELECT \*

FROM java.lang.Class c

WHERE c implements org.eclipse.mat.snapshot.model.IClass )

等同于

SELECT \* FROM ${snapshot}.getClasses()

* + - 1. 根据对象ID查询，多个ID逗号隔开

SELECT \* FROM 20815

SELECT \* FROM 20815,20975

* + - 1. 查询子类

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

等同于

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

SELECT v, v.@length FROM OBJECTS ( SELECT OBJECTS s.value FROM java.lang.String s ) v

### 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：

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

* + - 1. 例子

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. 例子

select \* from java.lang.String union (select \* from java.lang.StringBuilder)

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

## 获取属性

### 获取对象在堆中field：

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

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

### 获取对象内置JAVA相关属性：

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

例子：SELECT s.@usedHeapSize, s.@retainedHeapSize FROM java.lang.String s

| **Any heap object** | **IObject** | 内置属性 | **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. |

### 调取java 方法

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

例子 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 |

### 读取数组

* + - 1. 读取Object数组

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

* + - 1. 读取元数组

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

* + - 1. SELECT s.getValueAt(2) FROM int[] s WHERE (s.@length > 2)
      2. SELECT OBJECTS s[2] FROM java.lang.Object[] s
      3. SELECT OBJECTS s.getReferenceArray(2,1) FROM java.lang.Object[] s WHERE (s.@length > 2)
      4. SELECT OBJECTS s.@referenceArray.get(2) FROM java.lang.Object[] s WHERE (s.@length > 2)
      5. SELECT s.@GCRoots[2] FROM OBJECTS ${snapshot} s
      6. SELECT s.@GCRoots.get(2) FROM OBJECTS ${snapshot} s WHERE s.@GCRoots.@length > 2
      7. SELECT s.@GCRoots.subList(1,3)[1] FROM OBJECTS ${snapshot} s
      8. SELECT s.@GCRoots.subList(1,3).get(1) FROM OBJECTS ${snapshot} s
      9. 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. SELECT a[0] FROM java.util.ArrayList a
      2. SELECT a[0:-1] FROM java.util.ArrayList a
      3. SELECT h[0].@key, h[0].@value FROM java.util.HashMap h

### 内置OQL方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| toHex(number) | 转为16进制 |
| toString(object) | toString() |
| dominators(object) | 返回由Object直接控制的对象集合 |
| outbounds(object) | Returns the outbound referrer. |
| inbounds(object) | Returns the inbound referrer. |
| classof(object) | 获取当前对象的class |
| dominatorof(object) | 获取宿主对象, 如果-1，则表示没有 |

### 例子

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

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

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

## 模拟SQL中一些实现

* + - 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
         1. SELECT s.sz AS Size,

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

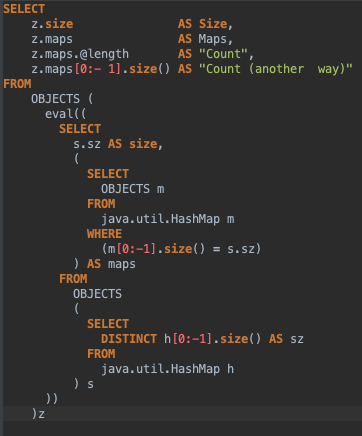
FROM OBJECTS ( SELECT DISTINCT h[0:-1].size() AS sz FROM java.util.HashMap h ) s

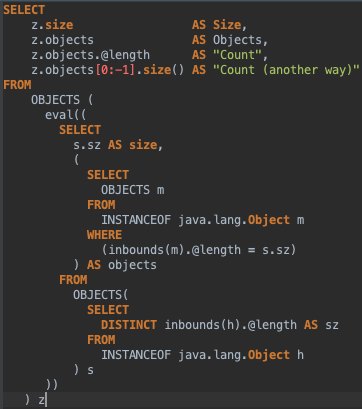
* + - * 1. SELECT s.sz AS Size,

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

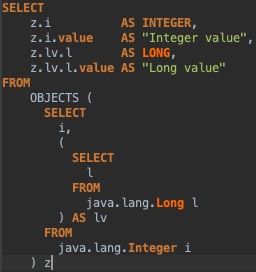
FROM OBJECTS ( SELECT DISTINCT inbounds(h).@length AS sz FROM INSTANCEOF java.lang.Object h ) s

* + - 1. COUNT

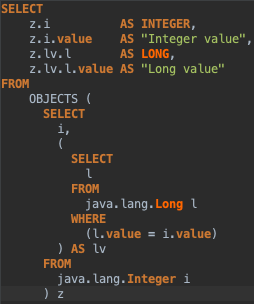




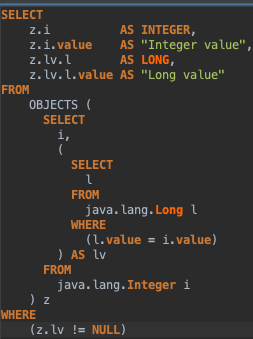
* + - 1. CROSS JOIN



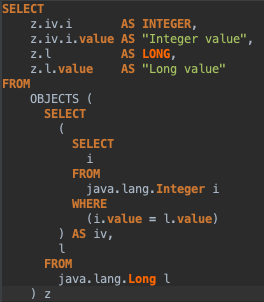
* + - 1. LEFT JOIN



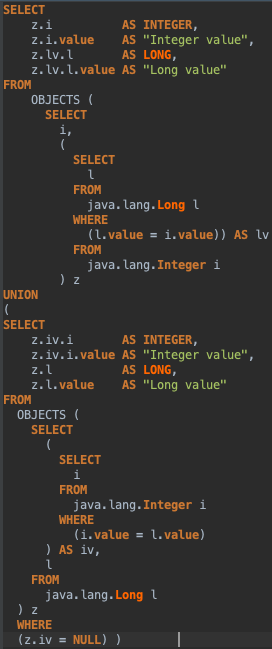
* + - 1. INNER JOIN



* + - 1. RIGHT JOIN



* + - 1. FULL OUTER JOIN



### 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

* + - 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

### select

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

### from

* + - 1. 多个对象，逗号隔开
      2. class name：java类的完全限定名或特定数组名或[L+ java类的完全限定名
         1. 注意：java类的完全限定名，当多个classloader时，不一定是唯一
         2. SELECT \* FROM java.lang.String
      3. identifier/object\_ID:class对象id
         1. SELECT \* FROM 0xe14a100
         2. SELECT \* FROM 0x2b7468c8,0x2b74aee0
         3. SELECT \* FROM 20815,20975
      4. regular
         1. SELECT \* FROM "java\.lang\..\*"
      5. OBJECTS 以对象的方式处理类
         1. SELECT \* FROM OBJECTS java.lang.String
      6. instanceof：表示也查询某个类的子类,如果不明确instanceof，则只精确查询class name指定的类的实例
         1. SELECT \* FROM INSTANCEOF java.lang.ref.Reference
      7. 子查询
         1. SELECT \* FROM ( SELECT \* FROM java.lang.Class c WHERE c IMPLEMENTS org.eclipse.mat.snapshot.model.IClass )

### where

* + - 1. 关系符：>=,<=,>,<,[NOT] LIKE,[NOT] IN,IMPLEMENTS,=,!=,AND,OR
      2. 例子：

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 s.count > 100 AND s.@retainedHeapSize > s.@usedHeapSize

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

SELECT \* FROM java.lang.String s WHERE s.@retainedHeapSize > 1024L

SELECT \* FROM java.lang.String s WHERE s.value != null AND s.value.@valueArray.@length >= 1 AND s.value.@valueArray.get(0) = 'j'

### 读取数组元素



## 内置对象

### 对象—heap

|  |  |
| --- | --- |
| 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  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 |
| head.livepaths(obj) | 查找对象活的引用链  select heap.livepaths(s) from java.lang.String s |
| heap.roots() | 获取堆的根对象枚举，每个root 对象都有以下属性：  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() | 等待垃圾收集的java对象的枚举 |

## 例子

* + - 1. select heap.findClass("java.lang.System").statics.props
      2. select heap.findClass("java.lang.String").fields.length
      3. select heap.findObject("0xf3800b58")
      4. select filter(heap.classes(), "/java.net./.test(it.name)")

## 对象函数

|  |  |
| --- | --- |
| 函数 | 说明 |
| classof(objname) | 返回指定对象的类  返回对象有以下属性：  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 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.  有以下方法：  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.  例子  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() |
| allocTrace(objName) | This returns allocation site trace of a given Java object if available.  返回对象的属性有：  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. |
| objectid(objName) | 获取对象的ID  select objectid(o) from java.lang.Object o |
| forEachReferrer(callback,obj) | 遍历指定对象的引用者 |
| reachables(obj[,omit]) | 获取指定对象的可达对象集合,忽略omit对象  select reachables(p) from java.util.Properties p  select reachables(u, 'java.net.URL.handler') from java.net.URL u |
| referrers(obj) | 返回指定对象的所有直接/间接引用者枚举  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 |
| referees(obj) | 返回指定对象的直接引用者集合  select referees(heap.findClass("java.io.File")) |
| refers(o1,o2) | 判断o1是否引用了o2 |
| root(obj) | 判断是否根对象，如果是则返回描述，否则返回null |
| sizeof(obj) | 对象大小(bytes)，但不包含其引用对象  select sizeof(o) from [I o |
| rsizeof(obj) | 对象大小，包含其引用对象,即堆深。其不仅与对象有关，还有当前对象的数据内容有关 |
| toHtml(obj) | 将对象转为html显示 |
| identical(o1,o2) | 判断o1,o2是否相同  select identical(heap.findClass("Foo").statics.bar, heap.findClass("AnotherClass").statics.bar) |
|  |  |

## 集合操作

|  |  |
| --- | --- |
| 函数 | 说明 |
| contains(objSet, booleanExpression) | 判断当前集合是否包含指定表达式的对象，  booleanExpression中内置对象有：  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated  例子：  select p from java.util.Properties p where contains(referrers(p), "classof(it).name == 'java.lang.Class'") |
| count(objSet, booleanExpression) | 当前集合包含指定表达式的对象的总数，  booleanExpression中内置对象有：  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated  例子  select count(heap.classes(), "/java.io./.test(it.name)") |
| filter(objSet, booleanExpression) | 获得当前集合包含指定表达式的对象的子集，  booleanExpression中内置对象有：  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated  例子  select filter(heap.classes(), "/java.io./.test(it.name)")  select filter(referrers(u), "! /java.net./.test(classof(it).name)") from java.net.URL u |
| length(objSet) | 返回集合/枚举中元素数量 |
| concat(objSet1,ObjSet2) | 链接两个集合/枚举 |
| map(objSet,transerRule) | 将集合中元素按特定规则转换  transerRule中内置对象：  it -> currently visited element  index -> index of the current element  array -> array/enumeration that is being iterated  result -> result array/enumeration  例子：  select map(heap.findClass("java.io.File").statics, "index + '=' + toHtml(it)") |
| max(setObj,[express]) | 获得集合中最大的元素，默认进行数值比较  express内置对象：  lhs -> left side element for comparison  rhs -> right side element for comparison  例子：  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') |
| min(setObj,[express]) | 最小值，其他同max  例子：  select min(heap.objects('java.util.Vector'), 'lhs.elementData.length < rhs.elementData.length') |
| sort(setObj,[express]) | 排序，其他同max  例子：  select sort(heap.objects('[C'), 'sizeof(lhs) - sizeof(rhs)')  select map(sort(heap.objects('[C'), 'sizeof(lhs) - sizeof(rhs)'), '{ size: sizeof(it), obj: it }') |
| top(set,expression,num) | 获取集合中，指定规则的头几个对象  内置对象：  lhs -> left side element for comparison  rhs -> right side element for comparison |
| sum(setObj,[express]) | 计算集合的累计值，默认进行数值计算  例子：  select sum(reachables(p), 'sizeof(it)') from java.util.Properties p |
| toArray(objSet) | 将集合转为数组 |
| unique(objSet) | 返回无重复的对象集合  例子：  select count(unique(map(heap.objects('java.lang.String'), 'it.value'))) |

## 内置方法

## 例子

* + - 1. 查询长度大于等于100的字符串:

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

* + - 1. 查询长度大于等于256的int数组:

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

* + - 1. 查询长度大于等于256的int数组:

select a from int[] a where a.length >= 256

* + - 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. })
    * + 1. 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';

}

} )

# 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

### 说明

创建Jar包，基于ZIP或ZLIB压缩格式

### 文档

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

### 基础指令

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

### 主要参数

|  |  |
| --- | --- |
| -c  --create | 创建归档 |
| -i=FILE  --generate-index=FILE | 产生索引文件 |
| -t  --list | 列出打包内容 |
| -u  --update | 更新覆盖已存在的包 |
| -x  --extract | 解压 |
| -d  --describe-module | 打印模块描述或名称 |
| -c DIR | 改变指定的目录，必须在命令行末尾 |
| -f=FILE  --file=FILE | 指定包名 |
| --release VERSION | 指定发布版本号，必须大于等于9的整数 |
| -v or --verbose | 打印全部信息 |
| 创建更新时参数 |  |
| -e=CLASSNAME  --main-class=CLASSNAME | 指定启动 |
| -m=FILE  --manifest=FILE | 指定清单信息文件 |
| -M  --no-manifest | 不创建清单文件 |
| --module-version=VERSION | 指定模块版本 |
| --hash-modules=PATTERN | 根据pattern计算哈希 |
| -p  --module-path | 指定生产哈希所依赖的模块 |
| @file | 从指定文件中加载参数 |
| -0  --no-compress | 生成不压缩的jar包 |

## 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

### 文档

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

### 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 |

# 可视化工具列表

## Visual VM

## JConsole

## Mission Control

## JHSDB