

JVM笔记无特殊说明，基于jdk8+

[零、参考资料 3](#_Toc28520456)

[一、常用JVM参数 3](#_Toc28520457)

[1. GC参数 3](#_Toc28520458)

[2. 堆参数 4](#_Toc28520459)

[3. 非堆参数 5](#_Toc28520460)

[4. 其他参数 5](#_Toc28520461)

[二、分析JVM 6](#_Toc28520462)

[1. JDK自带 6](#_Toc28520463)

[1.1. 基本工具 6](#_Toc28520464)

[1.2. 可视化工具 8](#_Toc28520465)

[2. MAT 9](#_Toc28520466)

[3. LINUX(centos7) 9](#_Toc28520467)

[3.1. top：实时显示系统各个进程的资源占用情况 9](#_Toc28520468)

[3.2. vmstat 10](#_Toc28520469)

[3.3. iostat 10](#_Toc28520470)

[3.4. pidstat 10](#_Toc28520471)

[4. Windows 10](#_Toc28520472)

[4.1. Perfmon 10](#_Toc28520473)

[4.2. Process Explorer 10](#_Toc28520474)

[4.3. pslist 10](#_Toc28520475)

[三、JVM常用执行指令 10](#_Toc28520476)

[1. 汇编基础 10](#_Toc28520477)

[2. 基础指令 10](#_Toc28520478)

[五、JVM基本原理 11](#_Toc28520479)

[1. JAVA技术体系 11](#_Toc28520480)

[2. 内存结构 12](#_Toc28520481)

[2.1. 程序计数器(Program Counter Register) 12](#_Toc28520482)

[2.2. JVM栈(JAVA Stack) 12](#_Toc28520483)

[2.3. 本地方法栈(Native Method Stack) 12](#_Toc28520484)

[2.4. 堆(HEAP) 12](#_Toc28520485)

[2.5. 方法区(Method Area) 12](#_Toc28520486)

[2.6. 运行时常量池(Run-Time Constant Pool) 12](#_Toc28520487)

[2.7. 直接内存(Direct Memory) 12](#_Toc28520488)

[2.8. 本地线程分配缓存(Thread Local Allocation Buffer, TLAB) 12](#_Toc28520489)

[2.9. 对象模型 12](#_Toc28520490)

[2.10. 对象定位 12](#_Toc28520491)

[3. 类结构、加载、执行 12](#_Toc28520492)

[4. 垃圾收集器内存分配和算法实现 12](#_Toc28520493)

[4.1. 判断对象是否活得 12](#_Toc28520494)

[4.2. 引用 13](#_Toc28520495)

[4.3. 垃圾收集算法 13](#_Toc28520496)

[5. 内存模型 13](#_Toc28520497)

[6. 编译原理 13](#_Toc28520498)

# 零、参考资料

Java虚拟机规范(Java SE 8版)

Java语言规范基于Java SE 8

深入理解Java虚拟机——JVM高级特性与最佳实践

实战Java虚拟机——JVM故障诊断与性能优化

垃圾回收的算法与实现

深入理解计算机系统

Stackoverflow:

https://stackoverflow.com/questions/16549066/java-major-and-minor-garbage-collections

OpenJdk：

https://openjdk.java.net/jeps/122

Oracle官方DOC：

https://docs.oracle.com/javase/8/javase-books.htm

https://docs.oracle.com/en/java/javase/13/

https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

https://www.oracle.com/webfolder/technetwork/tutorials/mooc/JVM\_Troubleshooting/week1/lesson1.pdf

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

http://cr.openjdk.java.net/~sundar/8022483/webrev.01/raw\_files/new/src/share/classes/com/sun/tools/hat/resources/oqlhelp.html

# 一、常用JVM参数

## GC参数

* 1. 打印GC基本日志

JDK8: -XX:PrintGC JDK9+: -Xlog:gc

* 1. 打印GC详细日志

JDK8: -XX:PrintGCDetails JDK9+: -Xlog:gc\*

-XX:PrintGCDetails 还会使JVM在退出前打印堆详细信息

* 1. 打印GC时堆的全面信息

JDK8: -XX:PrintHeapAtGC JDK9+:移除

* 1. 打印GC发生时间

JDK8: -XX:PrintGCTimeStamps JDK9+: -Xlog:gc\*已包含

* 1. 打印程序执行时间

-XX:+PrintGCApplicationConcurrentTime

* 1. 打印程序因STW而停顿的时间

-XX:+PrintGCApplicationStoppedTime

* 1. 打印软引用、弱引用、虚引用和Finallize队列信息

JDK8: +PrintReferenceGC JDK9+移除

* 1. 输出GC信息至文本

JDK8:-Xloggc:log/${filePath} JDK9+:-Xlog:gc:log/${filePath}

* 1. 打印类加载日志

JDK8: -XX:+TraceClassLoading JDK9+: -Xlog:class+load=info

* 1. 打印类卸载日志

JDK8: -XX:+TraceClassUnLoading JDK9+: -Xlog:class+unload=info

* 1. 显示当前的类信息柱状图

-XX:+PrintClassHistogram

* 1. 查看JVM运行显示参数

-XX:+PrintVMOptions

* 1. 打印JVM显示和隐式(默认)参数

-XX:+PrintCommandLineFlags

* 1. 打印所有系统参数

-XX:+PrintFlagsFinal

* 1. 在新生代和老年代使用串行回收器

-XX:UseSerialGC

* 1. 在新生代使用并行回收器

-XX:+UseParNewGC

* 1. 在老年代使用并行回收器

-XX:+UseParallelOldGC

* 1. 设置用于垃圾回收的线程数

-XX:ParallelGCThreads

* 1. 设置最大GC停顿时间

-XX:MaxGCPauseMillis

* 1. 设置吞吐量大小

-XX:GCTimeRatio

* 1. 打开开自适应GC策略

-XX:+UseAdaptiveSizePolicy

* 1. 使用G1

-XX:+UseG1GC

* 1. 设置停顿间隔

-XX:GCPauseIntervalMillis

* 1. 使用CMS

-XX:+UseConcMarkSweepGC

* 1. 设置CMS线程数

-XX:ParallelCMSThreads

* 1. 设置CMS触发时，老年代比例

-XX:CMSInitiatingOccupancyFraction

* 1. 设置CMS在FullGC后是否进行碎片整理

-XX:+UseCMSCompactAtFullCollection

* 1. 设置CMS进行内存压缩前，FullGC次数

-XX:CMSFullGCsBeforeCompaction

* 1. 允许对类元数据回收

-XX:+CMSClassUnloadingEnabled

* 1. 设置CMS触发时，永久区比例,仅在激活CMSClassUnloadingEnabled时有效

-XX:CMSInitiatingPermOccupancyFraction

* 1. 只有达到阈值才执行CMS GC

-XX:UseCMSInitiatingOccupancyOnly

## 堆参数

* 1. 初始堆空间

-Xms

* 1. 最大堆空间

-Xmx

* 1. 设置新生代大小

-Xmn

* 1. 设置新生代中eden区和from/to区的比例

-XX:SurvivorRatio

SurvivorRatio=eden/from=eden/to

* 1. 设置新生代与老年代的比例

-XX:NewRatio

NewRatio=老年代/新生代

* 1. 设置在内存溢出时到处整个堆信息

-XX:+HeapDumpOnOutOfMemoryError

* 1. 设置导出堆文件存放路径

-XX:HeapDumpPath

与-XX:+HeapDumpOnOutOfMemoryError配合

* 1. 设置大对象直接进入老年代的阈值

-XX:MaxTenuringThreshold

## 非堆参数

* 1. 最大直接内存

-XX:MaxDirectMemorySize

如果不设置，则默认与-Xmx相同

* 1. 每个线程的栈大小

-Xss

* 1. 打开TLAB

-XX:+UseTLAB

* 1. 打印TLAB相关信息

-XX:+PrintTLAB

JDK9+不支持

* 1. TLAB大小设置

-XX:TLABSize

* 1. 设置TLAB大小自动调整

-XX:+ResizeTLAB

## 其他参数

* 1. 设置工作模式为客户端

-client

* 1. 设置工作模式为服务端

-server

* 1. 禁用System.gc()（显示GC）

-XX:+DisableExplicitGC

* 1. 设置并发处理显示GC

-XX:+ExplicitGCInvokesConcurrent

* 1. 后台编译

-XX:+BackgroundCompilation

* 1. 开启逃逸分析

-XX:+DoEscapeAnalysis

仅在server模式下有效

* 1. 打开标量替换

-XX:+EliminateAllocation

* 1. 开启解释执行模式

-Xint

* 1. 开启编译模式

-Xcomp

* 1. 开启混合模式民，

-Xmixed

# 二、分析JVM

## JDK自带

### 基本工具

* + 1. jps

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

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jps.html#CHDCGECD

|  |  |
| --- | --- |
| -q | 只输出进程ID |
| -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 |
| -Joption | 将参数传递给java启动器，EX: jps -J-Xms48M -J-XX:+PrintGCDetails |

* + 1. jstat

查看java运行时信息

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jstat.html#BEHHGFAE

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

|  |  |
| --- | --- |
| -help | 帮助信息 |
| -class | 显示classLoader信息 |
| -compiler | 显示jit相关信息 |
| -gc | 显示gc信息 |
| -gccapacity | 显示各个代信息 |
| -gccause | 显示垃圾回收信息 |
| -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 | 显示垃圾回收信息，同gccause |
| -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 | 进程ID |
| interval | 周期时间 |
| count | 循环次数 |

* + 1. jinfo

查看java的参数，支持在运行时修改部分参数，这个命令在jdk8中处于实验性质

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jinfo.html#BCGEBFDD

jinfo [option] pid

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

* + 1. jmap

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

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jmap.html#CEGCECJB

jmap [options] pid

|  |  |
| --- | --- |
| 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:[live,] format=b, file=filename | 使用hprof二进制形式,输出jvm的heap内容到文件，live参数可选，如果指定，则只输出活对象 |
| -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 |

* + 1. jhat在jdk9+被移除，官方建议用visualVm代替
    2. jstack

导出线程栈信息，进行死锁检查

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jstack.html#BABGJDIF

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

* + 1. jstatd

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

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

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

* + 1. jcmd

将诊断命令请求发送到正在运行的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

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jcmd.html#CIHEEDIB

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

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

### 可视化工具

* + 1. Visual VM

|  |
| --- |
| 索引 |
| [OQL内置对象](#jvisualvmOQL内置对象) |
| [OQL对象函数](#jvisualvmOQL对象函数) |
| [OQL统计集合](#jvisualvmOQL统计集合) |
| [基本使用](#jvisualvm基本使用) |
| [bTrace插件](#bTrace插件) |
|  |
|  |

1. OQL(对象查询语言)

简介

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。OQL基于JavaScript表达式语言。

SELECT

select <JavaScript expression to select>

[ from [instanceof] <class name> <identifier>

[ where <JavaScript boolean expression to filter> ] ]

说明：

class name：java类的完全限定名，如：java.lang.String, java.util.ArrayList

instanceof：表示也查询某一个类的子类，如果不明确instanceof，则只精确查询class name指定的类

可以使用obj.field\_name语法访问Java字段，并且可以使用array [index]语法访问数组元素

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

2.查询长度大于等于256的int数组:

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

3.查询长度大于等于256的int数组:

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

4.显示所有File对象的文件路径:

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

5.选取所有的ClassLoader，包括子类:

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

6. 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)

7.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())

内置对象

|  |  |  |
| --- | --- | --- |
| 对象 | 说明 | 例子 |
| heap.forEachClass(callback) | 对每一个Class对象执行一个回调操作。其中 callback 为 Javascript 函数 | heap.forEachClass(function(it) {  if (classes[it.name] != null) {  if (multipleLoadedClasses[it.name] != null) {  multipleLoadedClasses[it.name] = multipleLoadedClasses[it.name] + 1;  } else {  multipleLoadedClasses[it.name] = 1;  }  } else {  classes[it.name] = it;  }  }); |
| heap.forEachObject (callback, clazz, includeSubtypes) | clazz:指定实例对象，默认为Java.lang.Object  includeSubtypes：是否包含子类，默认true |  |
| head.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. | select heap.findClass("java.util.Vector")  select heap.findClass("java.util.Vector").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集合 |  |
| heap.objects(clazz, [includeSubtypes], [filter]) | 返回堆快照中所有的对象的集合  clazz:指定类名称，默认java.lang.Object  includeSubtypes：是否包含子类，true  filter:过滤规则 | select heap.objects ("java.io.File",true, 'it.value !=null') |
| head.livepaths(obj) | 查找对象活的引用链 | select heap.livepaths(s) from java.lang.String s  输出例子：  java.lang.String#1600-> geym.zbase.ch7.heap.WebPage#57-> java.lang.Object[]#341-> java.util.Vector#11-> geym.zbase.ch7.heap.Student#3 |
| heap.roots() | 获取堆的根对象 |  |
| heap.finalizables() | 等待垃圾收集的java对象的枚举 |  |

对象函数

|  |  |  |
| --- | --- | --- |
| 函数 | 说明 | 例子 |
| 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. | select classof(o).name from instanceof java.lang.ref.Reference o |
| 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) | 获取指定对象的可达对象集合 | select reachables(p) from java.util.Properties p |
| referrers(obj) | 返回指定对象的引用者集合 | select count(referrers(o)) from java.lang.Object o |
| referees(obj) | 返回指定对象的直接引用者集合 |  |
| refers(firstobj,secondObj) | 判断firstobj是否引用了secondObj |  |
| 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，但不包含其引用对象 |  |
| 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 | select "<b>" + toHtml(o) + "</b>" from java.lang.Object o |
|  |  |  |
|  |  |  |

统计集合

|  |  |  |
| --- | --- | --- |
| 函数 | 说明 | 例子 |
| contains(objSet, booleanExpression) | 判断当前集合是否包含指定表达式的对象，  内置对象有：  **it** -> currently visited element  **index** -> index of the current element  **array** -> array/enumeration that is being iterated | select count(heap.classes(), "/java.io./.test(it.name)") |
| count(objSet, booleanExpression) | 当前集合包含指定表达式的对象的总数，  内置对象有：  **it** -> currently visited element  **index** -> index of the current element  **array** -> array/enumeration that is being iterated |  |
| filter(objSet, booleanExpression) | 获得当前集合包含指定表达式的对象的子集，  内置对象有：  **it** -> currently visited element  **index** -> index of the current element  **array** -> array/enumeration that is being iterated |  |
| length(objSet) | 返回集合中元素数量 |  |
| concat(objSet1,ObjSet2) | 链接两个集合 |  |
| map(objSet,transerRule) | 将集合中元素按特定规则转换  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 | 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  内置对象：  lhs -> left side element for comparison  rhs -> right side element for comparison | 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 |  |
| sort(setObj,[express]) | 排序，其他同max |  |
| 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 | select sum(map(reachables(p), 'sizeof(it)')) from java.util.Properties p  select sum(reachables(p), 'sizeof(it)') from java.util.Properties p |
| unique(objSet) | 返回无重复的对象集合 |  |

1. 基本使用

启动：

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a computer

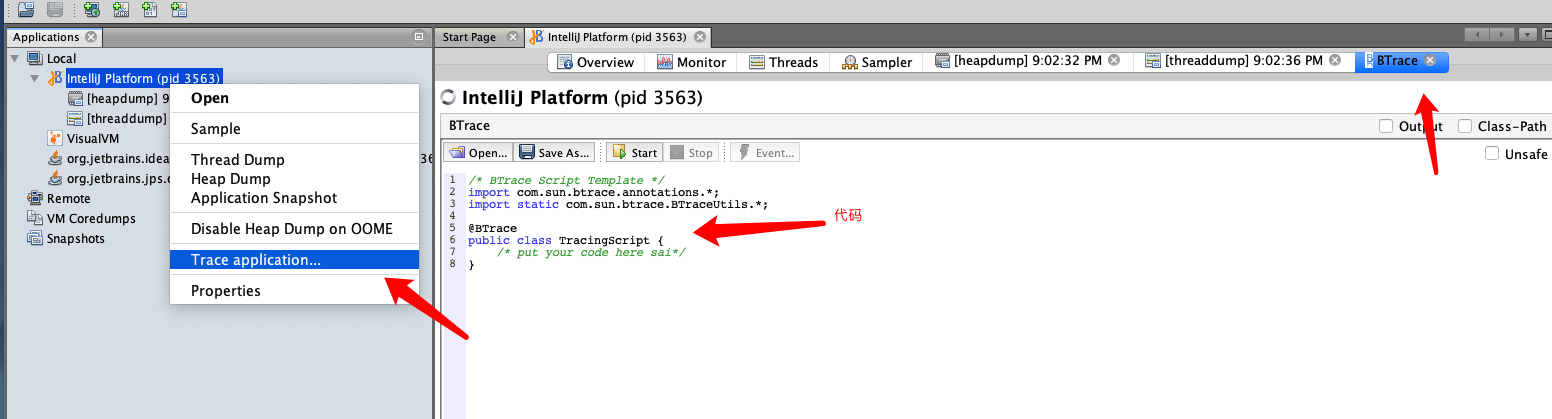
Description automatically generated

A screenshot of a social media post

Description automatically generated

1. BTrace

通过字节码注入，动态监控系统的运行情况。它可以跟踪指定的方法调用、构造函数调用和系统内存等信息。在Visual VM中安装插件BTrace后，右击Java程序打开Trece application



例子脚本

@TLS

private static long startTime = 0;

@OnMethod(clazz="/.+/", //监控任意类

method="/slowMethod/") //监控slowMethod方法

public static void startMethod(){

startTime = timeMillis();

}

@OnMethod(clazz="/.+/",method="/slowMethod/",location=@Location(Kind.RETURN))//方法返回时触发

public static void endMethod(){

long time = timeMillis() - startTime;

println(strcat("execute time(nanos): ", str(time)));

}

* + 1. JConsole
    2. Mission Control

## MAT

## LINUX(centos7)

### top：实时显示系统各个进程的资源占用情况



|  |  |  |
| --- | --- | --- |
| 第一部分 | | |
| 位置 | 参数 | 说明 |
| top | days | 系统已运行的时间 |
|  | users | 登陆用户数 |
|  | load average | 平均负载，1分钟，5分钟，15分钟 |
| Task | total | 进程总数 |
|  | running | 运行总数 |
|  | sleeping | 睡眠总数 |
|  | stopped | 停止总数 |
|  | zombie | 僵尸进程数(父进程已经退出,而该进程dead之后没有进程接受,就成为僵尸进程.(zombie)进程) |
| %Cpu | us | 用户占用率 |
|  | sy | 内核占用率 |
|  | ni | 用户改变过优先级的进程占用率 |
|  | id | 空闲CPU占用率 |
|  | wa | 等待中的CUP比例 |
|  | hi | 硬件中断请求占用率 |
|  | si | 软件中断请求占用率 |
|  | st | 被偷走百分比 |
| KiB Mem | total | 物理总量 |
|  | free | 空闲 |
|  | used | 使用 |
|  | buff/cache | 作为内核缓存的总量 |
| Kib Swap | total | 虚拟内存总量 |
|  | free | 空闲虚拟内存 |
|  | used | 虚拟内存使用量 |
|  | avail Mem | 可以用总量 |
| 第二部分 | | |
| head | PID | 进程号 |
|  | USER | 进程所有者用户 |
|  | PR | 优先级 |
|  | NI | nice值，负值表示高优先级，正值表示低优先级 |
|  | VIRT | 进程使用虚拟内存总量 |
|  | RES | 进程使用的、未被换出的物理内存大小 |
|  | SHR | 共享内存大小 |
|  | S | 进程状态。  D=不可中断的睡眠状态  R=运行  S=睡眠  T=跟踪/停止  Z=僵尸进程 |
|  | %CPU | CPU占比 |
|  | %MEM | 物理内存占比 |
|  | TIME+ | 使用CPU时间，单位 1/100秒 |
|  | COMMEND | 命令名/命令行 |

### ps

### vmstat

### iostat

### uptime

### dmesg

### mpstat

### sar

### free

### tcpdump

### netstat

### pidstat

### vim

### grep

### awk

## Windows

### Perfmon

### Process Explorer

### pslist

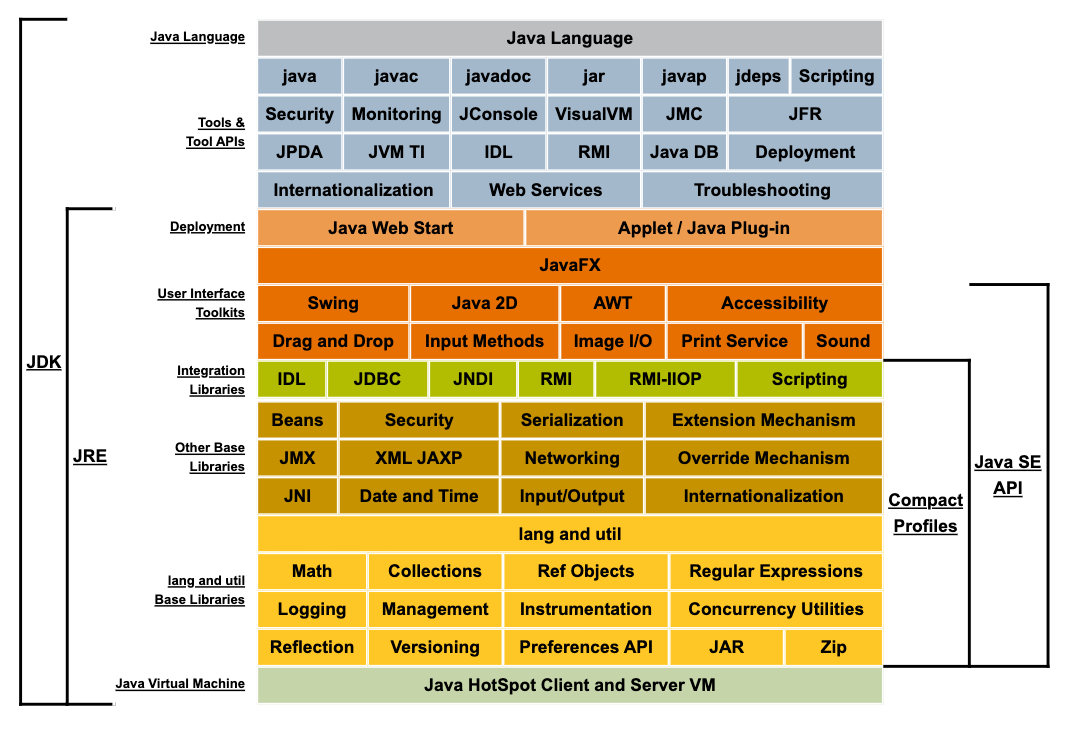
# 三、JVM常用执行指令

## 汇编基础

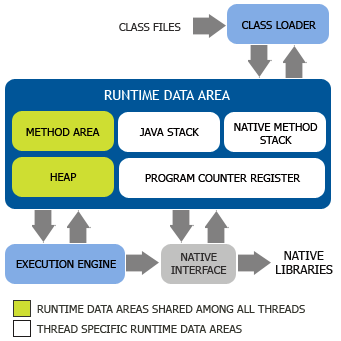
## 基础指令

# 五、JVM基本原理

## JAVA技术体系



## 内存结构



### 程序计数器(Program Counter Register)

### JVM栈(JAVA Stack)

### 本地方法栈(Native Method Stack)

### 堆(HEAP)

### 方法区(Method Area)

### 运行时常量池(Run-Time Constant Pool)

### 直接内存(Direct Memory)

### 本地线程分配缓存(Thread Local Allocation Buffer, TLAB)

### 对象模型

### 对象定位

## 类结构、加载、执行

## 垃圾收集器内存分配和算法实现

### 判断对象是否活得

* + 1. 引用计数法
       1. 概述
       2. 实现
       3. 优缺点
    2. 可达性分析算法
       1. 概述
       2. 实现
       3. 优缺点

### 引用

* + 1. 虚引用
    2. 弱引用
    3. 软引用
    4. 强引用
    5. finalize()

### 垃圾收集算法

* + 1. GC效率指标
    2. 标记-清除算法
    3. 复制算法
    4. 标记-压缩(Mark-Compact)算法
    5. 分代收集算法
    6. STW、安全点(Safepoint)、安全区域(Safe Region)
    7. 保守式GC
    8. 增量式GC
    9. RC Immix
  1. 垃圾收集器
     1. Serial
     2. ParNew
     3. Parallel Scavenge
     4. Serial Old
     5. Parallel Old
     6. CMS(Concurrent Mark Sweep)
     7. G1
     8. 内存分配与回收策略

## 内存模型

## 编译原理

堆分配参数示意图



Thread Local Allocation Buffer

逃逸分析(Escape Analysis)