目录

[jps:显示vm进程、启动参数 3](#_Toc466471900)

[null默认显示main方法simple name 3](#_Toc466471901)

[-m 显示附加参数,main参数 3](#_Toc466471902)

[-l (long) 显示main方法全路径 3](#_Toc466471903)

[-v (vm rags) 3](#_Toc466471904)

[组合 4](#_Toc466471905)

[-mlv 4](#_Toc466471906)

[jinfo:设置&启用禁用&打印 vm参数,打印系统参数 5](#_Toc466471907)

[null: 默认=打印vm参数和system变量 5](#_Toc466471908)

[-flags 5](#_Toc466471909)

[-sysprops 5](#_Toc466471910)

[-flag <name> jinfo -flag MaxHeapSize 3736 6](#_Toc466471911)

[-flag +|-<name> jinfo -flag +UseCompressedOops 3736 6](#_Toc466471912)

[-flag <name>=<value> jinfo -flag MaxHeapSize=777777777 3736 6](#_Toc466471913)

[jstat:每隔给定时间监控vm某些arg&统计量 7](#_Toc466471914)

[格式：jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]] 7](#_Toc466471915)

[-options 8](#_Toc466471916)

[-class Loaded Bytes Unloaded Bytes Time 8](#_Toc466471917)

[-compiler Compiled Failed Invalid Time FailedType FailedMethodsd 8](#_Toc466471918)

[-gc S0C(1st survivor capacity bytes) S1C(2nd survivor capacity bytes) S0U(1st survivor used space bytes) S1U(2st survivor used space bytes) EC(eden capacity bytes) EU(eden used space bytes) OC(old capacity bytes) OU(old used space bytes) MC(method area capacity bytes) MU(method area used space bytes) CCSC(compressed class space capacity bytes) CCSU(compressed class space used bytes) YGC(Young GC Count) YGCT(Young GC Time) FGC(Full GC Count) FGCT(Full GC Time) GCT(Total GC Time) 8](#_Toc466471919)

[-gccapacity NGCMN(new generation capacity min) NGCMX(~max) NGC(new generation capacity current) S0C() S1C() EC() OGCMN(old generation) OGCMX() OGC() OC() MCMN(metadata capacity min) MCMX() MC() CCSMN(compressed class size min) CCSMX CCSC() YGC() FGC() 8](#_Toc466471920)

[-gccause S0() S1() E() O() M() CCS() YGC() YGCT() FGC() FGCT GCT() LGCC() GCC() 9](#_Toc466471921)

[-gcmetacapacity MCMN(metadata capacity min) MCMX() MC(metadata capacity) CCSMN() CCSMX CCSC() YGC() FGC() FGCT() GCT() 9](#_Toc466471922)

[-gcnew S0C() S1C() S0U() S1U() TT(??) MTT(??) DSS(??) EC() EU YGC() YGCT() 9](#_Toc466471923)

[-gcnewcapacity NGCMN() NGCMX() NGC() S0CMX() S0C() S1CMX() S1C() ECMX() EC() YGC() FGC() 9](#_Toc466471924)

[-gcold MC(method area) MU() CCSC() CCSU() OC() OU() YGC() FGC() FGCT() GCT() 9](#_Toc466471925)

[-gcoldcapacity OGCMN() OGCMX() OGC() OC() YGC() FGC FGCT() GCT() 9](#_Toc466471926)

[-gcutil S0(space used percent) S1(space used percent) E(space used percent) O(space used percent) M(metadata space used percent) CCS(space used percent) YGC(gc count) YGCT(gc time) FGC(gc count) FGCT(time)GCT(time) 9](#_Toc466471927)

[-printcompilation ( JIT ?)Compiled Size Type Method 9](#_Toc466471928)

[jmap:查看堆内存对象&空间&配置&统计信息 10](#_Toc466471929)

[null:内存信息 10](#_Toc466471930)

[-heap 查看堆配置&空间(New(eden,survivor),Old)使用情况 10](#_Toc466471931)

[-histo[:live] 打印java对象堆柱状图 11](#_Toc466471932)

[-clstats 打印classloader统计数据 12](#_Toc466471933)

[-finalizerinfo 打印当前等待finalization的对象数目 12](#_Toc466471934)

[-dump jmap -dump:live,format=b,file=x.bin 3736 dump java heap in hprof binary format 常导出来用jhat查看 12](#_Toc466471935)

[jhat:用来查看jmap -dump出来的 bin文件，会启动一个web服务，默认http://localhost:7000/,jhat x.bin 13](#_Toc466471936)

[-port default 7000,set port 13](#_Toc466471937)

[javap: 14](#_Toc466471938)

[null:反编译并显示类下方法 14](#_Toc466471939)

[-classpath 指定自定义加载类路径 15](#_Toc466471940)

[-l line&local var table 15](#_Toc466471941)

[-public(only)|package(default public&protected&package)|protected(public&protected)|private(all) 显示级别 16](#_Toc466471942)

[-s 输出了内部类型、方法签名 17](#_Toc466471943)

[-sysinfo 输出最后修改日期、md5 17](#_Toc466471944)

# jps:显示vm进程、启动参数

## null默认显示main方法simple name

6260 Jps

3736

7912 Launcher

524 Bootstrap

-q (queit)

1904

3736

7912

524

## -m 显示附加参数,main参数

8132 Jps -m

3736

524 Bootstrap start

## -l (long) 显示main方法全路径

6704 sun.tools.jps.Jps

3736

7912 org.jetbrains.jps.cmdline.Launcher

524 org.apache.catalina.startup.Bootstrap

## -v (vm rags)

4420 Jps -Dapplication.home=C:\Program Files\Java\jdk1.8.0 -Xms8m

3736 -Xms128m -Xmx750m -XX:ReservedCodeCacheSize=240m -XX:+UseConcMarkSweepGC -

XX:SoftRefLRUPolicyMSPerMB=50 -ea -Dsun.io.useCanonCaches=false -Djava.net.prefe

## 组合

### -mlv

# jinfo:设置&启用禁用&打印 vm参数,打印系统参数

## null: 默认=打印vm参数和system变量

=

-flags

+

-sysprops

## -flags

Attaching to process ID 3736, please wait...

Debugger attached successfully.

Server compiler detected.

JVM version is 25.0-b70

Non-default VM flags: -XX:ErrorFile=null -XX:+HeapDumpOnOutOfMemoryError -XX:Hea

pDumpPath=null -XX:InitialHeapSize=134217728 -XX:MaxHeapSize=786432000 -XX:MaxNe

wSize=262144000 -XX:MaxTenuringThreshold=6 -XX:MinHeapDeltaBytes=196608 -XX:NewS

ize=1310720 -XX:OldPLABSize=16 -XX:OldSize=132907008 -XX:-OmitStackTraceInFastTh

row -XX:ReservedCodeCacheSize=251658240 -XX:SoftRefLRUPolicyMSPerMB=50 -XX:+UseC

ompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:-UseLa

rgePagesIndividualAllocation -XX:+UseParNewGC

Command line: -Xms128m -Xmx750m -XX:ReservedCodeCacheSize=240m -XX:+UseConcMark

SweepGC -XX:SoftRefLRUPolicyMSPerMB=50 -ea -Dsun.io.useCanonCaches=false -Djava.

net.preferIPv4Stack=true -XX:+HeapDumpOnOutOfMemoryError -XX:-OmitStackTraceInFa

stThrow -Djb.vmOptionsFile=E:\IntelliJ IDEA 2016.2.4\bin\idea64.exe.vmoptions -X

bootclasspath/a:E:\IntelliJ IDEA 2016.2.4\lib\boot.jar -Didea.paths.selector=Int

elliJIdea2016.2 -Didea.jre.check=true -XX:ErrorFile=C:\Users\DELL-13\java\_error\_

in\_idea\_%p.log -XX:HeapDumpPath=C:\Users\DELL-13\java\_error\_in\_idea.hprof

## -sysprops

Attaching to process ID 3736, please wait...

Debugger attached successfully.

Server compiler detected.

JVM version is 25.0-b70

java.vendor = Oracle Corporation

sun.management.compiler = HotSpot 64-Bit Tiered Compilers

sun.nio.ch.bugLevel =

idea.paths.selector = IntelliJIdea2016.2

jnidispatch.path = C:\Users\DELL-13\.IntelliJIdea2016.2\system\tmp\jna2023092355

827809894.dll

winp.unpack.dll.to.parent.dir = false

os.name = Windows 7

## -flag <name> jinfo -flag MaxHeapSize 3736

-XX:MaxHeapSize=786432000

## -flag +|-<name> jinfo -flag +UseCompressedOops 3736

## -flag <name>=<value> jinfo -flag MaxHeapSize=777777777 3736

# jstat:每隔给定时间监控vm某些arg&统计量

## 格式：jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]

jstat -gc 3736 1000 3 每隔1s打印 一共只打印3行

jstat -gc 3736 2s 3 每隔2s打印 一共只打印3行

jstat -gc -h2 3736 1000每隔1s打印，每2行显示一次title行

jstat -gc -t -h1 3736 2000 -t另外显示时间戳 8595.3 单位s

## -options

### -class Loaded Bytes Unloaded Bytes Time

### -compiler Compiled Failed Invalid Time FailedType FailedMethodsd

### -gc S0C(1st survivor capacity bytes) S1C(2nd survivor capacity bytes) S0U(1st survivor used space bytes) S1U(2st survivor used space bytes) EC(eden capacity bytes) EU(eden used space bytes) OC(old capacity bytes) OU(old used space bytes) MC(method area capacity bytes) MU(method area used space bytes) CCSC(compressed class space capacity bytes) CCSU(compressed class space used bytes) YGC(Young GC Count) YGCT(Young GC Time) FGC(Full GC Count) FGCT(Full GC Time) GCT(Total GC Time)

### -gccapacity NGCMN(new generation capacity min) NGCMX(~max) NGC(new generation capacity current) S0C() S1C() EC() OGCMN(old generation) OGCMX() OGC() OC() MCMN(metadata capacity min) MCMX() MC() CCSMN(compressed class size min) CCSMX CCSC() YGC() FGC()

### -gccause S0() S1() E() O() M() CCS() YGC() YGCT() FGC() FGCT GCT() LGCC() GCC()

### -gcmetacapacity MCMN(metadata capacity min) MCMX() MC(metadata capacity) CCSMN() CCSMX CCSC() YGC() FGC() FGCT() GCT()

### -gcnew S0C() S1C() S0U() S1U() TT(??) MTT(??) DSS(??) EC() EU YGC() YGCT()

### -gcnewcapacity NGCMN() NGCMX() NGC() S0CMX() S0C() S1CMX() S1C() ECMX() EC() YGC() FGC()

### -gcold MC(method area) MU() CCSC() CCSU() OC() OU() YGC() FGC() FGCT() GCT()

### -gcoldcapacity OGCMN() OGCMX() OGC() OC() YGC() FGC FGCT() GCT()

### -gcutil S0(space used percent) S1(space used percent) E(space used percent) O(space used percent) M(metadata space used percent) CCS(space used percent) YGC(gc count) YGCT(gc time) FGC(gc count) FGCT(time)GCT(time)

### -printcompilation ( JIT ?)Compiled Size Type Method

# jmap:查看堆内存对象&空间&配置&统计信息

## null:内存信息

Attaching to process ID 3736, please wait...

Debugger attached successfully.

Server compiler detected.

JVM version is 25.0-b70

0x0000000019700000 72K E:\IntelliJ IDEA 2016.2.4\bin\focuskiller64.dll

0x0000000050ce0000 260K C:\Program Files\Java\jdk1.8.0\jre\bin\t2k.dll

0x0000000050d30000 252K C:\Program Files\Java\jdk1.8.0\jre\bin\splashscr

een.dll

0x0000000051010000 1624K C:\Program Files\Java\jdk1.8.0\jre\bin\awt.dll

## -heap 查看堆配置&空间(New(eden,survivor),Old)使用情况

Attaching to process ID 3736, please wait...

Debugger attached successfully.

Server compiler detected.

JVM version is 25.0-b70

using parallel threads in the new generation.

using thread-local object allocation.

Concurrent Mark-Sweep GC

Heap Configuration:

MinHeapFreeRatio = 40

MaxHeapFreeRatio = 70

MaxHeapSize = 786432000 (750.0MB)

NewSize = 1310720 (1.25MB)

MaxNewSize = 262144000 (250.0MB)

OldSize = 132907008 (126.75MB)

NewRatio = 2

SurvivorRatio = 8

MetaspaceSize = 21807104 (20.796875MB)

CompressedClassSpaceSize = 1073741824 (1024.0MB)

MaxMetaspaceSize = 17592186044415 MB

G1HeapRegionSize = 0 (0.0MB)

Heap Usage:

New Generation (Eden + 1 Survivor Space):

capacity = 173539328 (165.5MB)

used = 97842920 (93.3102798461914MB)

free = 75696408 (72.1897201538086MB)

56.3808337439223% used

Eden Space:

capacity = 154271744 (147.125MB)

used = 94932768 (90.53494262695312MB)

free = 59338976 (56.590057373046875MB)

61.5360697549384% used

From Space:

capacity = 19267584 (18.375MB)

used = 2910152 (2.7753372192382812MB)

free = 16357432 (15.599662780761719MB)

15.103876023065476% used

To Space:

capacity = 19267584 (18.375MB)

used = 0 (0.0MB)

free = 19267584 (18.375MB)

0.0% used

concurrent mark-sweep generation:

capacity = 385257472 (367.41015625MB)

used = 7106562446934514688 (6.777346083578601E12MB)

free = 10814839961204 MB

1.8446267661057886E12% used

72661 interned Strings occupying 7498024 bytes.

## -histo[:live] 打印java对象堆柱状图

num #instances #bytes class name

----------------------------------------------

1: 291364 29966320 [C

2: 56914 15131200 [I

3: 56379 11287984 [B

4: 286930 6886320 java.lang.String

5: 91840 6518944 [Ljava.lang.Object;

6: 46531 4744432 java.lang.Class

7: 52484 1679488 java.util.HashMap$Node

8: 49983 1599456 org.jdom.Attribute

9: 35988 1439520 java.util.LinkedHashMap$Entry

10: 9164 1267648 [Ljava.util.HashMap$Node;

## -clstats 打印classloader统计数据

## -finalizerinfo 打印当前等待finalization的对象数目

Attaching to process ID 3736, please wait...

Debugger attached successfully.

Server compiler detected.

JVM version is 25.0-b70

Number of objects pending for finalization: 0

## -dump jmap -dump:live,format=b,file=x.bin 3736 dump java heap in hprof binary format 常导出来用jhat查看

# jhat:用来查看jmap -dump出来的 bin文件，会启动一个web服务，默认<http://localhost:7000/>,jhat x.bin

## -port default 7000,set port

# javap:

javap java.util.List

## null:反编译并显示类下方法

Compiled from "List.java"

public interface java.util.List<E> extends java.util.Collection<E> {

public abstract int size();

public abstract boolean isEmpty();

public abstract boolean contains(java.lang.Object);

public abstract java.util.Iterator<E> iterator();

public abstract java.lang.Object[] toArray();

public abstract <T> T[] toArray(T[]);

public abstract boolean add(E);

public abstract boolean remove(java.lang.Object);

public abstract boolean containsAll(java.util.Collection<?>);

public abstract boolean addAll(java.util.Collection<? extends E>);

public abstract boolean addAll(int, java.util.Collection<? extends E>);

public abstract boolean removeAll(java.util.Collection<?>);

public abstract boolean retainAll(java.util.Collection<?>);

public void replaceAll(java.util.function.UnaryOperator<E>);

public void sort(java.util.Comparator<? super E>);

public abstract void clear();

public abstract boolean equals(java.lang.Object);

public abstract int hashCode();

public abstract E get(int);

public abstract E set(int, E);

public abstract void add(int, E);

public abstract E remove(int);

public abstract int indexOf(java.lang.Object);

public abstract int lastIndexOf(java.lang.Object);

public abstract java.util.ListIterator<E> listIterator();

public abstract java.util.ListIterator<E> listIterator(int);

public abstract java.util.List<E> subList(int, int);

public java.util.Spliterator<E> spliterator();

}

## -classpath | cp 指定自定义加载类路径

C:\Users\DELL-13\Desktop>javap -classpath D:\algorithem\Common\target\classes co

m.jiangli.common.utils.SpringUtil

## -l line&local var table

Compiled from "SpringUtil.java"

public class com.jiangli.common.utils.SpringUtil implements org.springframework.

context.ApplicationContextAware {

public com.jiangli.common.utils.SpringUtil();

LineNumberTable:

line 14: 0

LocalVariableTable:

Start Length Slot Name Signature

0 5 0 this Lcom/jiangli/common/utils/SpringUtil;

public final void setApplicationContext(org.springframework.context.Applicatio

nContext) throws org.springframework.beans.BeansException;

LineNumberTable:

line 19: 0

line 20: 4

LocalVariableTable:

Start Length Slot Name Signature

0 5 0 this Lcom/jiangli/common/utils/SpringUtil;

0 5 1 applicationContext Lorg/springframework/context/Appl

icationContext;

public static org.springframework.context.ApplicationContext getApplicationCon

text();

LineNumberTable:

line 23: 0

public static java.lang.Object getBean(java.lang.String);

LineNumberTable:

line 27: 0

LocalVariableTable:

Start Length Slot Name Signature

0 10 0 beanName Ljava/lang/String;

}

## -public(only)|package(default public&protected&package)|protected(public&protected)|private(all) 显示级别

-c decompile

Compiled from "SpringUtil.java"

public class com.jiangli.common.utils.SpringUtil implements org.springframework.

context.ApplicationContextAware {

public com.jiangli.common.utils.SpringUtil();

Code:

0: aload\_0

1: invokespecial #1 // Method java/lang/Object."<init>":

()V

4: return

public final void setApplicationContext(org.springframework.context.Applicatio

nContext) throws org.springframework.beans.BeansException;

Code:

0: aload\_1

1: putstatic #2 // Field applicationContext:Lorg/spr

ingframework/context/ApplicationContext;

4: return

public static org.springframework.context.ApplicationContext getApplicationCon

text();

Code:

0: getstatic #2 // Field applicationContext:Lorg/spr

ingframework/context/ApplicationContext;

3: areturn

public static java.lang.Object getBean(java.lang.String);

Code:

0: getstatic #2 // Field applicationContext:Lorg/spr

ingframework/context/ApplicationContext;

3: aload\_0

4: invokeinterface #3, 2 // InterfaceMethod org/springframewo

rk/context/ApplicationContext.getBean:(Ljava/lang/String;)Ljava/lang/Object;

9: areturn

}

## -s 输出了内部类型、方法签名

Compiled from "SpringUtil.java"

public class com.jiangli.common.utils.SpringUtil implements org.springframework.

context.ApplicationContextAware {

public com.jiangli.common.utils.SpringUtil();

descriptor: ()V

public final void setApplicationContext(org.springframework.context.Applicatio

nContext) throws org.springframework.beans.BeansException;

descriptor: (Lorg/springframework/context/ApplicationContext;)V

public static org.springframework.context.ApplicationContext getApplicationCon

text();

descriptor: ()Lorg/springframework/context/ApplicationContext;

public static java.lang.Object getBean(java.lang.String);

descriptor: (Ljava/lang/String;)Ljava/lang/Object;

}

## -sysinfo 输出最后修改日期、md5

Classfile /D:/algorithem/Common/target/classes/com/jiangli/common/utils/SpringUt

il.class

Last modified 2016-11-1; size 1006 bytes

MD5 checksum 636e20971e10e503ce5001480a49f10b

Compiled from "SpringUtil.java"

public class com.jiangli.common.utils.SpringUtil implements org.springframework.

context.ApplicationContextAware {

public com.jiangli.common.utils.SpringUtil();

public final void setApplicationContext(org.springframework.context.Applicatio

nContext) throws org.springframework.beans.BeansException;

public static org.springframework.context.ApplicationContext getApplicationCon

text();

public static java.lang.Object getBean(java.lang.String);

}

# apt: 仅存在jdk1\_6(Annotation processing tool) 是一种处理注释的工具,它对源代码文件进行检测找出其中的Annotation，使用Annotation进行额外的处理。

**取代 APT 的是更优的 JSR 269 API - "Pluggable Annotation Processing API".**

编译程序源代码时，生成附属文件

编写Annotation处理器需要使用JDK-lib目录中的tools.jar 里的如下4个包。  
com.sun.mirror.apt: //和APT交互的接口   
com.sun.mirror.declaration: //包含各种封装类成员类方法类声明的接口。   
com.sun.mirror.type: //包含各种封装源代码中程序元素的接口。   
com.sun.mirror.util: //提供了用于处理类型和声明的一些工具。

每个Annotation处理器需要实现com.sun.mirror.apt包下的AnnotationProcessor接口，这个接口中定义了一个"process"方法，该方法是由apt调用Annotation处理器时将被用到的。

一个Annotation处理器可以处理一种或多种Annotation类型。

1. 通常情况下Annotation处理器实例是由其相应的工厂返回Annotation处理器工厂应该实现AnnotationProcessorFactory接口APT将调用工厂类的getProcessorFor方法来获得Annotation处理器。   
2. 在调用过程中APT将提供给工厂类一个AnnotationProcessorEnvironment对象。  
3. AnnotationProcessorEnvironment对象是APT工具与注释环境通信的途径。

public class HibernateAnnotationFactory implements AnnotationProcessorFactory {

 // 所有支持的注释类型

 public Collection<String> supportedAnnotationTypes() {

  return Arrays.asList("Property", "IdProperty", "Persistent");

 }

 // 返回所有支持的选项

 public Collection<String> supportedOptions() {

  return Arrays.asList(new String[0]);

 }

 // 返回Annotation处理器

 public AnnotationProcessor getProcessorFor( Set<AnnotationTypeDeclaration> atds, AnnotationProcessorEnvironment env) {

  return new HibernateAnnotationProcessor(env);

 }

}

public class HibernateAnnotationProcessor implements AnnotationProcessor {

 // Annotation处理器环境是该处理器与APT交互的重要途径

 private AnnotationProcessorEnvironment env;

 // 构造HibernateAnnotationProcessor对象时获得处理器环境

 public HibernateAnnotationProcessor(AnnotationProcessorEnvironment env) {

  this.env = env;

 }

 // 循环处理每个对象

 public void process() {

}

apt -factory HibernateAnnotationFactory Person.java

<http://www.javacui.com/Theory/367.html>

apt -cp D:\algorithem\JDK\target\classes -factorypath D:\algorithem\JDK\target\classes -factory com.jiangli.jdk.tool.apt.StaticsFactory -cp D:\algorithem\JDK\src\main\java -nocompile \*.java

## jdb:java debugger,调试工具