**代码清单1-1 演示代码**

**public class JVMDemoTest {**

**/\*\***

**\* 获取当前jvm的内存信息**

**\* @return**

**\*/**

**public static String toMemoryInfo() {**

**Runtime runtime = Runtime.getRuntime ();**

**int freeMemory = ( int ) (runtime.freeMemory() / 1024 / 1024);**

**int totalMemory = ( int ) (runtime.totalMemory() / 1024 / 1024);**

**return freeMemory + "M/" + totalMemory + "M(free/total)" ;**

**}**

**/\*\***

**\***

**\* @param args**

**\*/**

**public static void main(String[] args) {**

**System.out.println( "memory info :" + toMemoryInfo ());**

**}**

**}**

**代码清单1-2 清单1-1运行输出（设置参数）**

**memory info :60M/61M(free/total)**

**代码清单1-3 清单1-1运行输出（未设置参数）**

**memory info :56M/57M(free/total)**

**代码清单1-4 Linux配置参数示例**

**在/usr/local/apache-tomcat-5.5.23/bin 目录下的catalina.sh**

**添加:JAVA\_OPTS='-Xms512m -Xmx1024m'**

**或者 JAVA\_OPTS="-server -Xms800m -Xmx800m**

**-XX:MaxNewSize=256m" 或者 CATALINA\_OPTS="-server -Xms256m -Xmx300m"**

**代码清单1-5 内存泄漏演示代码**

**import Java.util.HashMap;**

**import Java.util.Map;**

**public class MemoryLeakDemo {**

**static class Key {**

**Integer id;**

**Key(Integer id) {**

**this.id = id;**

**}**

**@Override**

**public int hashCode() {**

**return id.hashCode();**

**}**

**}**

**public static void main(String[] args) {**

**Map m = new HashMap();**

**while (true)**

**for (int i = 0; i < 10000; i++)**

**if (!m.containsKey(new Key(i)))**

**m.put(new Key(i), "Number:" + i);**

**}**

**}**

**代码清单1-6 内存泄漏演示代码运行输出**

**Exception in thread "main" Java.lang.OutOfMemoryError: GC overhead limit exceeded**

**at Java.util.HashMap.newNode(Unknown Source)**

**at Java.util.HashMap.putVal(Unknown Source)**

**at Java.util.HashMap.put(Unknown Source)**

**at MemoryLeakDemo.main(MemoryLeakDemo.Java:23)**

**代码清单1-7 StringIntern实例程序**

**public class StringInternDemo {**

**public static void main(String[] args) {**

**String s = new String("1");**

**s.intern();**

**String s2 = "1";**

**System.out.println(s == s2);**

**String s3 = new String("1") + new String("1");**

**s3.intern();**

**String s4 = "11";**

**System.out.println(s3 == s4);**

**}**

**}**

**代码清单1-8 LinkedList数据结构源码**

**Entry<E> {**

**E element;**

**Entry<E> previous;**

**Entry<E> next;**

**}**

**代码清单1-9 运行Java版本及输出**

**C:\Users\Administrator>Java -version**

**Java version "1.8.0\_101"**

**Java(TM) SE Runtime Environment (build 1.8.0\_101-b13)**

**Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)**

**代码清单1-10 启用编译模式**

**C:\Users\Administrator>Java -Xcomp -version**

**Java version "1.8.0\_101"**

**Java(TM) SE Runtime Environment (build 1.8.0\_101-b13)**

**Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, compiled mode)**

**代码清单1-11 两种模式对比程序**

**public class JVMDemoTest {**

**/\*\***

**\* 获取当前jvm的内存信息**

**\* @return**

**\*/**

**public static String toMemoryInfo() {**

**int freeMemory;**

**int totalMemory;**

**String calcResult = null;**

**Runtime runtime = Runtime.getRuntime ();**

**for(int i = 0;i<100000;i++){**

**freeMemory = ( int ) (runtime.freeMemory() / 1024 / 1024);**

**totalMemory = ( int ) (runtime.totalMemory() / 1024 / 1024);**

**calcResult = freeMemory + "M/" + totalMemory + "M(free/total)";**

**}**

**return calcResult ;**

**}**

**/\*\***

**\***

**\* @param args**

**\*/**

**public static void main(String[] args) {**

**long b = System.currentTimeMillis();**

**System.out.println( "memory info :" + toMemoryInfo ());**

**System.out.println(System.currentTimeMillis() - b);**

**}**

**}**

**代码清单1-12 C++代码**

**Student \*pStudent = new cmBaseSchoolStudent();**

**If(pStudent->Register(kDestory)!=NO\_ERROR)**

**Delete pStudent;**

**代码清单1-13 吞吐量示例**

**public class PigInThePython {**

**static volatile List pigs = new ArrayList();**

**static volatile int pigsEaten = 0;**

**static final int ENOUGH\_PIGS = 5000;**

**public static void main(String[] args) throws InterruptedException {**

**new PigEater().start();**

**new PigDigester().start();**

**}**

**static class PigEater extends Thread {**

**@Override**

**public void run() {**

**while (true) {**

**pigs.add(new byte[32 \* 1024 \* 1024]); //32MB per pig**

**if (pigsEaten > ENOUGH\_PIGS) return;**

**takeANap(100);**

**}**

**}**

**}**

**static class PigDigester extends Thread {**

**@Override**

**public void run() {**

**long start = System.currentTimeMillis();**

**while (true) {**

**takeANap(2000);**

**pigsEaten+=pigs.size();**

**pigs = new ArrayList();**

**if (pigsEaten > ENOUGH\_PIGS) {**

**System.out.format("Digested %d pigs in %d ms.%n",pigsEaten, System.currentTimeMillis()-start);**

**return;**

**}**

**}**

**}**

**}**

**static void takeANap(int ms) {**

**try {**

**Thread.sleep(ms);**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**代码清单1-14 吞吐量示例**

**Attaching to process ID \*\*\*\*, please wait...**

**Debugger attached successfully.**

**Client compiler detected.**

**JVM version is 1.8.0\_101**

**using thread-local object allocation.**

**Mark Sweep Compact GC**

**Heap Configuration:**

**MinHeapFreeRatio = 40**

**MaxHeapFreeRatio = 70**

**MaxHeapSize = 67108864 (64.0MB)**

**NewSize = 655360 (0.625MB)**

**MaxNewSize = 4294901760 (4095.9375MB)**

**OldSize = 1441792 (1.375MB)**

**NewRatio = 12**

**SurvivorRatio = 8**

**PermSize = 8388608 (8.0MB)**

**MaxPermSize = 67108864 (64.0MB)**

**Heap Usage:**

**New Generation (Eden + 1 Survivor Space):**

**capacity = 4521984 (4.3125MB)**

**used = 1510200 (1.4402389526367188MB)**

**free = 3011784 (2.8722610473632812MB)**

**33.39684527853261% used**

**Eden Space:**

**capacity = 4063232 (3.875MB)**

**used = 1495992 (1.4266891479492188MB)**

**free = 2567240 (2.4483108520507812MB)**

**36.81778446320565% used**

**From Space:**

**capacity = 458752 (0.4375MB)**

**used = 14208 (0.0135498046875MB)**

**free = 444544 (0.4239501953125MB)**

**3.0970982142857144% used**

**To Space:**

**capacity = 458752 (0.4375MB)**

**used = 0 (0.0MB)**

**free = 458752 (0.4375MB)**

**0.0% used**

**tenured generation:**

**capacity = 59342848 (56.59375MB)**

**used = 36321192 (34.638587951660156MB)**

**free = 23021656 (21.955162048339844MB)**

**61.20567721994064% used**

**Perm Generation:**

**capacity = 11796480 (11.25MB)**

**used = 11712040 (11.169471740722656MB)**

**free = 84440 (0.08052825927734375MB)**

**99.28419325086806% used**

**代码清单1-15　Stop-the-World示例**

**import Java.util.HashMap;**

**public class StopWorldDemo {**

**public static class MyThread extends Thread{**

**HashMap map = new HashMap();**

**public void run(){**

**try{**

**while(true){**

**if(map.size()\*512/1024/1024>=400){**

**map.clear();//防止内存溢出**

**System.out.println("clean map");**

**}**

**byte[] b1;**

**for(int i=0;i<100;i++){**

**b1 = new byte[512];//模拟内存占用**

**map.put(System.nanoTime(),b1);**

**}**

**}**

**}catch(Exception ex){**

**ex.printStackTrace();**

**}**

**}**

**}**

**public static class PrintThread extends Thread{**

**public final long starttime = System.currentTimeMillis();**

**public void run(){**

**try{**

**while(true){**

**//每毫秒打印时间信息**

**long t = System.currentTimeMillis()-starttime;**

**System.out.println(t/1000+"."+t%1000);**

**Thread.sleep(1000);**

**}**

**}catch(Exception ex){**

**ex.printStackTrace();**

**}**

**}**

**}**

**public static void main(String[] args){**

**MyThread t = new MyThread();**

**PrintThread p = new PrintThread();**

**t.start();**

**p.start();**

**}**

**}**

**代码清单1-16　Stop-the-World示例运行输出**

**0.0**

**1.14**

**clean map**

**2.29**

**3.309**

**clean map**

**4.324**

**clean map**

**5.807**

**clean map**

**6.978**

**clean map**

**7.993**

**clean map**

**9.8**

**clean map**

**10.148**

**11.522**

**clean map**

**12.755**

**clean map**

**代码清单1-17　Stop-the-World示例的GC输出**

**0.366: [GC 0.366: [DefNew: 139776K->17472K(157248K), 0.2675607 secs] 139776K->127626K(506816K), 0.2677332 secs] [Times: user=0.19 sys=0.08, real=0.26 secs]**

**0.788: [GC 0.788: [DefNew: 157248K->17471K(157248K), 0.3404208 secs] 267402K->261455K(506816K), 0.3405538 secs] [Times: user=0.31 sys=0.03, real=0.34 secs]**

**1.241: [GC 1.241: [DefNew: 157247K->157247K(157248K), 0.0000146 secs]1.241: [Tenured: 243983K->349567K(349568K), 0.4988513 secs] 401231K->393834K(506816K), [Perm : 380K->380K(12288K)], 0.4990866 secs] [Times: user=0.50 sys=0.00, real=0.50 secs]**

**1.940: [Full GC 1.940: [Tenured: 349567K->47656K(349568K), 0.1760066 secs] 506815K->47656K(506816K), [Perm : 380K->380K(12288K)], 0.1761751 secs] [Times: user=0.17 sys=0.00, real=0.17 secs]**

**2.226: [GC 2.227: [DefNew: 139776K->17472K(157248K), 0.2130968 secs] 187432K->185248K(506816K), 0.2132255 secs] [Times: user=0.22 sys=0.00, real=0.22 secs]**

**2.555: [GC 2.555: [DefNew: 157248K->17471K(157248K), 0.2630955 secs] 325024K->323914K(506816K), 0.2632084 secs] [Times: user=0.27 sys=0.00, real=0.27 secs]**

**2.922: [GC 2.922: [DefNew: 157247K->157247K(157248K), 0.0000138 secs]2.922: [Tenured: 306443K->349567K(349568K), 0.5437268 secs] 463690K->455014K(506816K), [Perm : 380K->380K(12288K)], 0.5439029 secs] [Times: user=0.53 sys=0.00, real=0.53 secs]**

**3.556: [Full GC 3.556: [Tenured: 349567K->51995K(349568K), 0.1789923 secs] 506815K->51995K(506816K), [Perm : 380K->379K(12288K)], 0.1791206 secs] [Times: user=0.19 sys=0.00, real=0.19 secs]**

**3.845: [GC 3.845: [DefNew: 139776K->17471K(157248K), 0.2132299 secs] 191771K->190545K(506816K), 0.2133538 secs] [Times: user=0.20 sys=0.00, real=0.20 secs]**

**4.174: [GC 4.174: [DefNew: 157247K->17471K(157248K), 0.2663958 secs] 330321K->329202K(506816K), 0.2665229 secs] [Times: user=0.27 sys=0.00, real=0.26 secs]**

**4.569: [GC 4.569: [DefNew: 157247K->157247K(157248K), 0.0000158 secs]4.569: [Tenured: 311730K->12917K(349568K), 0.1204363 secs] 468978K->12917K(506816K), [Perm : 379K->379K(12288K)], 0.1206203 secs] [Times: user=0.13 sys=0.00, real=0.13 secs]**

**4.798: [GC 4.798: [DefNew: 139776K->17471K(157248K), 0.2108548 secs] 152693K->151356K(506816K), 0.2109783 secs] [Times: user=0.20 sys=0.00, real=0.20 secs]**

**5.122: [GC 5.122: [DefNew: 157247K->17472K(157248K), 0.2431866 secs] 291132K->289949K(506816K), 0.2432983 secs] [Times: user=0.25 sys=0.00, real=0.25 secs]**

**5.467: [GC 5.467: [DefNew: 157248K->157248K(157248K), 0.0000134 secs]5.467: [Tenured: 272477K->349567K(349568K), 0.4970060 secs] 429725K->420767K(506816K), [Perm : 379K->379K(12288K)], 0.4971599 secs] [Times: user=0.48 sys=0.00, real=0.48 secs]**

**6.058: [Full GC 6.058: [Tenured: 349567K->41509K(349568K), 0.1328209 secs] 506815K->41509K(506816K), [Perm : 379K->379K(12288K)], 0.1329192 secs] [Times: user=0.14 sys=0.00, real=0.14 secs]**

**6.274: [GC 6.274: [DefNew: 139776K->17472K(157248K), 0.1368488 secs] 181285K->154007K(506816K), 0.1369451 secs] [Times: user=0.14 sys=0.00, real=0.14 secs]**

**6.495: [GC 6.495: [DefNew: 157248K->17471K(157248K), 0.1743854 secs] 293783K->268532K(506816K), 0.1744881 secs] [Times: user=0.19 sys=0.00, real=0.19 secs]**

**6.760: [GC 6.760: [DefNew: 157247K->157247K(157248K), 0.0000126 secs]6.760: [Tenured: 251060K->349567K(349568K), 0.3610367 secs] 408308K->388050K(506816K), [Perm : 379K->379K(12288K)], 0.3611760 secs] [Times: user=0.37 sys=0.00, real=0.37 secs]**

**7.227: [Full GC 7.227: [Tenured: 349567K->33158K(349568K), 0.1144929 secs] 506815K->33158K(506816K), [Perm : 379K->379K(12288K)], 0.1145849 secs] [Times: user=0.13 sys=0.00, real=0.13 secs]**

**7.414: [GC 7.414: [DefNew: 139776K->17471K(157248K), 0.1112127 secs] 172934K->132730K(506816K), 0.1113000 secs] [Times: user=0.11 sys=0.00, real=0.11 secs]**

**7.599: [GC 7.599: [DefNew: 157247K->17472K(157248K), 0.1426927 secs] 272506K->234470K(506816K), 0.1427827 secs] [Times: user=0.14 sys=0.00, real=0.14 secs]**

**7.823: [GC 7.823: [DefNew: 157248K->157248K(157248K), 0.0000107 secs]7.823: [Tenured: 216998K->340621K(349568K), 0.3163073 secs] 374246K->340621K(506816K), [Perm : 379K->379K(12288K)], 0.3164363 secs] [Times: user=0.31 sys=0.00, real=0.31 secs]**

**8.217: [GC 8.217: [DefNew: 139776K->139776K(157248K), 0.0000111 secs]8.217: [Tenured: 340621K->349567K(349568K), 0.3631919 secs] 480397K->440326K(506816K), [Perm : 379K->379K(12288K)], 0.3633182 secs] [Times: user=0.36 sys=0.00, real=0.36 secs]**

**8.643: [Full GC 8.643: [Tenured: 349567K->37244K(349568K), 0.1115731 secs] 506815K->37244K(506816K), [Perm : 379K->379K(12288K)], 0.1116607 secs] [Times: user=0.11 sys=0.00, real=0.11 secs]**

**8.821: [GC 8.821: [DefNew: 139776K->17472K(157248K), 0.0952736 secs] 177020K->127601K(506816K), 0.0953525 secs] [Times: user=0.11 sys=0.00, real=0.11 secs]**

**8.983: [GC 8.983: [DefNew: 157248K->17471K(157248K), 0.1197464 secs] 267377K->219190K(506816K), 0.1198281 secs] [Times: user=0.11 sys=0.00, real=0.11 secs]**

**9.171: [GC 9.171: [DefNew: 157247K->17472K(157248K), 0.1685964 secs] 358966K->312990K(506816K), 0.1686990 secs] [Times: user=0.17 sys=0.00, real=0.17 secs]**

**9.467: [GC 9.467: [DefNew: 157248K->157248K(157248K), 0.0000150 secs]9.467: [Tenured: 295518K->349567K(349568K), 0.5542238 secs] 452766K->452375K(506816K), [Perm : 379K->379K(12288K)], 0.5544066 secs] [Times: user=0.55 sys=0.00, real=0.55 secs]**

**10.113: [Full GC 10.114: [Tenured: 349567K->52002K(349568K), 0.1808479 secs] 506815K->52002K(506816K), [Perm : 379K->379K(12288K)], 0.1809750 secs] [Times: user=0.19 sys=0.00, real=0.19 secs]**

**10.407: [GC 10.407: [DefNew: 139776K->17472K(157248K), 0.2176954 secs] 191778K->190499K(506816K), 0.2178194 secs] [Times: user=0.22 sys=0.00, real=0.22 secs]**

**10.740: [GC 10.740: [DefNew: 157248K->17471K(157248K), 0.2684413 secs] 330275K->329159K(506816K), 0.2685582 secs] [Times: user=0.28 sys=0.00, real=0.28 secs]**

**11.122: [GC 11.122: [DefNew: 157247K->157247K(157248K), 0.0000134 secs]11.122: [Tenured: 311687K->349567K(349568K), 0.5415732 secs] 468935K->462097K(506816K), [Perm : 379K->379K(12288K)], 0.5417346 secs] [Times: user=0.50 sys=0.00, real=0.55 secs]**

**11.757: [Full GC 11.757: [Tenured: 349567K->48817K(349568K), 0.1491027 secs] 506815K->48817K(506816K), [Perm : 379K->379K(12288K)], 0.1492014 secs] [Times: user=0.16 sys=0.00, real=0.16 secs]**

**11.990: [GC 11.990: [DefNew: 139776K->17472K(157248K), 0.1398503 secs] 188593K->162017K(506816K), 0.1399383 secs] [Times: user=0.14 sys=0.00, real=0.14 secs]**

**12.253: [GC 12.253: [DefNew: 157248K->17471K(157248K), 0.2044428 secs] 301793K->293736K(506816K), 0.2045332 secs] [Times: user=0.20 sys=0.00, real=0.20 secs]**

**12.539: [GC 12.539: [DefNew: 157247K->157247K(157248K), 0.0000114 secs]12.539: [Tenured: 276264K->349567K(349568K), 0.3627016 secs] 433512K->403701K(506816K), [Perm : 379K->379K(12288K)], 0.3628280 secs] [Times: user=0.36 sys=0.00, real=0.36 secs]**

**12.986: [Full GC 12.986: [Tenured: 349567K->27685K(349568K), 0.1033825 secs] 506815K->27685K(506816K), [Perm : 379K->379K(12288K)], 0.1034725 secs] [Times: user=0.11 sys=0.00, real=0.11 secs]**

**13.154: [GC 13.154: [DefNew: 139776K->17472K(157248K), 0.0961716 secs] 167461K->116529K(506816K), 0.0962454 secs]**

**代码清单1-18 System.gc示例**

**class TestGC**

**{**

**public static void main(String[] args)**

**{**

**new TestGC();**

**System.gc();**

**System.runFinalization();**

**}**

**}**

**代码清单1-19 Runtime.gc()源代码**

**JVM\_ENTRY\_NO\_ENV(void,JVM\_GC(void))**

**JVMWrapper(“JVM\_GC”);**

**If(!DisableExplicitGC){**

**Universe:heap()->collect(GCCause::\_Java\_lang\_system\_gc);**

**}**

**JVM\_END**

**代码清单1-20 强制使用System.gc后清单1-18运行输出**

**[Full GC (System.gc()) 908K->526K(8192K), 0.0250794 secs]**

[Eden: 1024.0K(7168.0K)->0.0B(4096.0K) Survivors: 0.0B->0.0B Heap: 908.8K(60.0M)->526.7K(8192.0K)], [Metaspace: 2551K->2551K(1056768K)]

[Times: user=0.00 sys=0.00, real=0.02 secs]

Heap

garbage-first heap total 8192K, used 526K [0x00000000c4200000, 0x00000000c4300040, 0x0000000100000000)

region size 1024K, 1 young (1024K), 0 survivors (0K)

Metaspace used 2561K, capacity 4486K, committed 4864K, reserved 1056768K

**class space used 286K, capacity 386K, committed 512K, reserved 1048576K**

**代码清单1-21 开启选项-XX:+ExplicitGCInvokesConcurrent后清单1-18运行输出**

**[GC pause (System.gc()) (young) (initial-mark), 0.0018781 secs]**

[Parallel Time: 1.6 ms, GC Workers: 2]

[GC Worker Start (ms): Min: 397.4, Avg: 397.5, Max: 397.5, Diff: 0.0]

[Ext Root Scanning (ms): Min: 0.5, Avg: 0.5, Max: 0.5, Diff: 0.0, Sum: 0.9]

[Update RS (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]

[Processed Buffers: Min: 0, Avg: 0.0, Max: 0, Diff: 0, Sum: 0]

[Scan RS (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]

[Code Root Scanning (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]

[Object Copy (ms): Min: 1.0, Avg: 1.0, Max: 1.1, Diff: 0.0, Sum: 2.1]

[Termination (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]

[Termination Attempts: Min: 1, Avg: 1.0, Max: 1, Diff: 0, Sum: 2]

[GC Worker Other (ms): Min: 0.0, Avg: 0.1, Max: 0.1, Diff: 0.0, Sum: 0.1]

[GC Worker Total (ms): Min: 1.6, Avg: 1.6, Max: 1.6, Diff: 0.0, Sum: 3.2]

[GC Worker End (ms): Min: 399.0, Avg: 399.0, Max: 399.0, Diff: 0.0]

[Code Root Fixup: 0.0 ms]

[Code Root Purge: 0.0 ms]

[Clear CT: 0.0 ms]

[Other: 0.2 ms]

[Choose CSet: 0.0 ms]

[Ref Proc: 0.1 ms]

[Ref Enq: 0.0 ms]

[Redirty Cards: 0.0 ms]

[Humongous Register: 0.0 ms]

[Humongous Reclaim: 0.0 ms]

[Free CSet: 0.0 ms]

[Eden: 1024.0K(7168.0K)->0.0B(5120.0K) Survivors: 0.0B->1024.0K Heap: 908.8K(60.0M)->608.1K(60.0M)]

[Times: user=0.02 sys=0.00, real=0.00 secs]

[GC concurrent-root-region-scan-start]

[GC concurrent-root-region-scan-end, 0.0006240 secs]

[GC concurrent-mark-start]

[GC concurrent-mark-end, 0.0001534 secs]

[GC remark [Finalize Marking, 0.0000274 secs] [GC ref-proc, 0.0000553 secs] [Unloading, 0.0016140 secs], 0.0018180 secs]

[Times: user=0.00 sys=0.00, real=0.00 secs]

[GC cleanup 608K->608K(60M), 0.0002289 secs]

[Times: user=0.00 sys=0.00, real=0.00 secs]

Heap

garbage-first heap total 61440K, used 608K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)

region size 1024K, 2 young (2048K), 1 survivors (1024K)

Metaspace used 2561K, capacity 4486K, committed 4864K, reserved 1056768K

**class space used 286K, capacity 386K, committed 512K, reserved 1048576K**

**代码清单1-22 混合GC日志范例**

**[G1Ergonomics (Mixed GCs) continue mixed GCs, reason: candidate old regions available, candidate old regions: 363 regions, reclaimable: 9830652576 bytes (10.40 %), threshold: 10.00 %]**

[Parallel Time: 145.1 ms, GC Workers: 23]

[GC Worker Start (ms): Min: 251176.0, Avg: 251176.4, Max: 251176.7, Diff: 0.7]

[Ext Root Scanning (ms): Min: 0.8, Avg: 1.2, Max: 1.7, Diff: 0.9, Sum: 28.1]

[Update RS (ms): Min: 0.0, Avg: 0.3, Max: 0.6, Diff: 0.6, Sum: 5.8]

[Processed Buffers: Min: 0, Avg: 1.6, Max: 9, Diff: 9, Sum: 37]

[Scan RS (ms): Min: 6.0, Avg: 6.2, Max: 6.3, Diff: 0.3, Sum: 143.0]

[Object Copy (ms): Min: 136.2, Avg: 136.3, Max: 136.4, Diff: 0.3, Sum: 3133.9]

[Termination (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.3]

[GC Worker Other (ms): Min: 0.0, Avg: 0.1, Max: 0.2, Diff: 0.2, Sum: 1.9]

[GC Worker Total (ms): Min: 143.7, Avg: 144.0, Max: 144.5, Diff: 0.8, Sum: 3313.0]

[GC Worker End (ms): Min: 251320.4, Avg: 251320.5, Max: 251320.6, Diff: 0.2]

[Code Root Fixup: 0.0 ms]

[Clear CT: 6.6 ms]

[Other: 26.8 ms]

[Choose CSet: 0.2 ms]

[Ref Proc: 16.6 ms]

[Ref Enq: 0.9 ms]

[Free CSet: 2.0 ms]

[Eden: 3904.0M(3904.0M)->0.0B(4448.0M) Survivors: 576.0M->32.0M Heap: 63.7G(88.0G)->58.3G(88.0G)]

**[Times: user=3.43 sys=0.01, real=0.18 secs]**

代码清单2-2 虚拟机栈示例代码 TestJVMStack

**public class TestJVMStack {**

**private int count = 0;**

**//没有出口的递归函数**

**public void recursion(){**

**count++;//每次调用深度加1**

**recursion();//递归**

**}**

**public void testStack(){**

**try{**

**recursion();**

**}catch(Throwable e){**

**System.out.println("deep of stack is "+count);//打印栈溢出的深度**

**e.printStackTrace();**

**}**

**}**

**public static void main(String[] args){**

**TestStack ts = new TestStack();**

**ts.testStack();**

**}**

**}**

代码清单2-3 代码2-2程序运行输出

**Java.lang.StackOverflowError**

**at TestStack.recursion(TestStack.Java:7)**

**at TestStack.recursion(TestStack.Java:7)**

**at TestStack.recursion(TestStack.Java:7)**

**at TestStack.recursion(TestStack.Java:7)**

**at TestStack.recursion(TestStack.Java:7)**

**at TestStack.recursion(TestStack.Java:7)**

**at TestStack.recursion(TestStack.Java:7)deep of stack is 9013**

代码清单2-4 虚拟机栈示例代码 TestJVMStack1

**public class TestJVMStack1 {**

**private int count = 0;**

**//没有出口的递归函数**

**public void recursion(long a,long b,long c) throws InterruptedException{**

**long d=0,e=0,f=0;**

**count++;//每次调用深度加1**

**recursion(a,b,c);//递归**

**}**

**public void testStack(){**

**try{**

**recursion(1L,2L,3L);**

**}catch(Throwable e){**

**System.out.println("deep of stack is "+count);//打印栈溢出的深度**

**e.printStackTrace();**

**}**

**}**

**public static void main(String[] args){**

**TestStack ts = new TestStack();**

**ts.testStack();**

**}**

**}**

代码清单2-5 代码2-4运行输出

**deep of stack is 3432**

**Java.lang.StackOverflowError**

**at TestStack.recursion(TestStack.Java:8)**

代码清单2-6 虚拟机栈示例代码 testGC

**public class testGC {**

**public static void test1()**

**{**

**byte[] a = new byte[6\*1024\*1024];**

**}**

**System.gc();**

**System.out.println("first explict gc over");**

**public static void main(String[] args){**

**testGC.test1();**

**}**

**}**

代码清单2-7 逃逸分析示例代码 escapeAnalysisClass

**public class escapeAnalysisClass{**

**public static B b;**

**public void globalVariablePointerEscape(){//给全局变量赋值，发生逃逸**

**b=new B();**

**}**

**public B methodPointerEscape(){//方法返回值，发生逃逸**

**return new B();**

**}**

**public void instancePassPointerEscape(){**

**methodPointerEscape().printClassName(this);//实例引用发生逃逸**

**}**

**}**

**class B{**

**public void printClassName(G g){**

**System.out.println(g.getClass().getName());**

**}**

**}**

**public class G {**

**public static B b;**

**public void globalVariablePointerEscape(){//给全局变量赋值，发生逃逸**

**b=new B();**

**}**

**public B methodPointerEscape(){//方法返回值，发生逃逸**

**return new B();**

**}**

**public void instancePassPointerEscape(){**

**methodPointerEscape().printClassName(this);//实例引用发生逃逸**

**}**

**}**

代码清单2-8 逃逸分析优化示例代码 my\_method

**public void my\_method(){**

**V v=new V();**

**//use v**

**......**

**v=null;**

**}**

代码清单2-9 堆分配示例 TestHeapGC

**public class TestHeapGC {**

**public static void main(String[] args){**

**byte[] b1 = new byte[1024\*1024/2];**

**byte[] b2 = new byte[1024\*1024\*8];**

**b2 = null;**

**b2 = new byte[1024\*1024\*8];//进行一次年轻代GC**

**System.gc();**

**}**

**}**

代码清单2-10 JVM选项

**-XX:+PrintGCDetails -XX:SurvivorRatio=8 -XX:MaxTenuringThreshold=15 -Xms40M -Xmx40M -Xmn20M**

代码清单2-11 2-9代码运行后GC输出

**[GC [DefNew: 9031K->661K(18432K), 0.0022784 secs] 9031K->661K(38912K), 0.0023178 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]**

**Heap**

**def new generation total 18432K, used 9508K [0x34810000, 0x35c10000, 0x35c10000)**

**eden space 16384K, 54% used [0x34810000, 0x350b3e58, 0x35810000)**

**from space 2048K, 32% used [0x35a10000, 0x35ab5490, 0x35c10000)**

**to space 2048K, 0% used [0x35810000, 0x35810000, 0x35a10000)**

**tenured generation total 20480K, used 0K [0x35c10000, 0x37010000, 0x37010000)**

**the space 20480K, 0% used [0x35c10000, 0x35c10000, 0x35c10200, 0x37010000)**

**compacting perm gen total 12288K, used 374K [0x37010000, 0x37c10000, 0x3b010000)**

**the space 12288K, 3% used [0x37010000, 0x3706db10, 0x3706dc00, 0x37c10000)**

**ro space 10240K, 51% used [0x3b010000, 0x3b543000, 0x3b543000, 0x3ba10000)**

**rw space 12288K, 55% used [0x3ba10000, 0x3c0ae4f8, 0x3c0ae600, 0x3c610000)**

代码清单2-12 2-9代码运行后GC输出

**[GC [DefNew: 9031K->661K(18432K), 0.0023186 secs] 9031K->661K(38912K), 0.0023597 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]**

**[Full GC (System) [Tenured: 0K->8853K(20480K), 0.0179368 secs] 9180K->8853K(38912K), [Perm : 374K->374K(12288K)], 0.0179893 secs] [Times: user=0.00 sys=0.02, real=0.02 secs]**

**Heap**

**def new generation total 18432K, used 327K [0x34810000, 0x35c10000, 0x35c10000)**

**eden space 16384K, 2% used [0x34810000, 0x34861f28, 0x35810000)**

**from space 2048K, 0% used [0x35a10000, 0x35a10000, 0x35c10000)**

**to space 2048K, 0% used [0x35810000, 0x35810000, 0x35a10000)**

**tenured generation total 20480K, used 8853K [0x35c10000, 0x37010000, 0x37010000)**

**the space 20480K, 43% used [0x35c10000, 0x364b5458, 0x364b5600, 0x37010000)**

**compacting perm gen total 12288K, used 374K [0x37010000, 0x37c10000, 0x3b010000)**

**the space 12288K, 3% used [0x37010000, 0x3706db40, 0x3706dc00, 0x37c10000)**

**ro space 10240K, 51% used [0x3b010000, 0x3b543000, 0x3b543000, 0x3ba10000)**

**rw space 12288K, 55% used [0x3ba10000, 0x3c0ae4f8, 0x3c0ae600, 0x3c610000)**

代码清单2-13 回收永久区示例 permGenGC

**public class permGenGC {**

**public static void main(String[] args){**

**for(int i=0;i<Integer.MAX\_VALUE;i++){**

**String t = String.valueOf(i).intern();//加入常量池**

**}**

**}**

**}**

代码清单2-14 JVM设置

**-XX:PermSize=2M -XX:MaxPermSize=4M -XX:+PrintGCDetails**

代码清单2-15 GC输出日志

**[Full GC [Tenured: 0K->149K(10944K), 0.0177107 secs] 3990K->149K(15872K), [Perm : 4096K->374K(4096K)], 0.0181540 secs] [Times: user=0.02 sys=0.02, real=0.03 secs]**

**[Full GC [Tenured: 149K->149K(10944K), 0.0165517 secs] 3994K->149K(15936K), [Perm : 4096K->374K(4096K)], 0.0169260 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]**

**[Full GC [Tenured: 149K->149K(10944K), 0.0166528 secs] 3876K->149K(15936K), [Perm : 4096K->374K(4096K)], 0.0170333 secs] [Times: user=0.02 sys=0.00, real=0.01 secs]**

**代码清单2-16逃脱回收实验**

**public class FinalizeEscapeGC {**

**public static FinalizeEscapeGC SAVE\_HOOK = null;**

**public void isAlive() {**

**System.out.println("yes, i am still alive");**

**}**

**@Override**

**protected void finalize() throws Throwable {**

**super.finalize();**

**System.out.println("finalize mehtod executed!");**

**FinalizeEscapeGC.SAVE\_HOOK = this;**

**}**

**public static void main(String[] args) throws Throwable {**

**SAVE\_HOOK = new FinalizeEscapeGC();**

**//对象第一次成功拯救自己**

**SAVE\_HOOK = null;**

**System.gc();**

**// 因为Finalizer方法优先级很低，暂停0.5秒，以等待它**

**Thread.sleep(500);**

**if (SAVE\_HOOK != null) {**

**SAVE\_HOOK.isAlive();**

**} else {**

**System.out.println("no, i am dead");**

**}**

**// 下面这段代码与上面的完全相同，但是这次自救却失败了**

**SAVE\_HOOK = null;**

**System.gc();**

**// 因为Finalizer方法优先级很低，暂停0.5秒，以等待它**

**Thread.sleep(500);**

**if (SAVE\_HOOK != null) {**

**SAVE\_HOOK.isAlive();**

**} else {**

**System.out.println("no, i am dead");**

**}**

**}**

**}**

**代码清单2-17逃脱回收实验运行输出**

**finalize mehtod executed!**

**yes, i am still alive**

**no, i am dead**

**代码清单2-18年轻代串行收集器工作输出**

**[GC [DefNew: 3468K->150K(9216K), 0.0028638 secs][Tenured: 1562K->1712K(10240K), 0.0084220 secs] 3468K->1712K(19456K), [Perm : 377K->377K(12288K)], 0.0113816 secs] [Times: user=0.02 sys=0.00, real=0.01 secs]**

**代码清单2-19使用-XX:+UseSerialGC参数**

**Heap**

**def new generation total 4928K, used 1373K [0x27010000, 0x27560000, 0x2c560000)**

**eden space 4416K, 31% used [0x27010000, 0x27167628, 0x27460000)**

**from space 512K, 0% used [0x27460000, 0x27460000, 0x274e0000)**

**to space 512K, 0% used [0x274e0000, 0x274e0000, 0x27560000)**

**tenured generation total 10944K, used 0K [0x2c560000, 0x2d010000, 0x37010000)**

**the space 10944K, 0% used [0x2c560000, 0x2c560000, 0x2c560200, 0x2d010000)**

**compacting perm gen total 12288K, used 376K [0x37010000, 0x37c10000, 0x3b010000)**

**the space 12288K, 3% used [0x37010000, 0x3706e0b8, 0x3706e200, 0x37c10000)**

**ro space 10240K, 51% used [0x3b010000, 0x3b543000, 0x3b543000, 0x3ba10000)**

**rw space 12288K, 55% used [0x3ba10000, 0x3c0ae4f8, 0x3c0ae600, 0x3c610000)**

**代码清单2-20使用-XX:+UseParNewGC参数**

**Heap**

**par new generation total 4928K, used 1373K [0x0f010000, 0x0f560000, 0x14560000)**

**eden space 4416K, 31% used [0x0f010000, 0x0f167620, 0x0f460000)**

**from space 512K, 0% used [0x0f460000, 0x0f460000, 0x0f4e0000)**

**to space 512K, 0% used [0x0f4e0000, 0x0f4e0000, 0x0f560000)**

**tenured generation total 10944K, used 0K [0x14560000, 0x15010000, 0x1f010000)**

**the space 10944K, 0% used [0x14560000, 0x14560000, 0x14560200, 0x15010000)**

**compacting perm gen total 12288K, used 2056K [0x1f010000, 0x1fc10000, 0x23010000)**

**the space 12288K, 16% used [0x1f010000, 0x1f2121d0, 0x1f212200, 0x1fc10000)**

**No shared spaces configured.**

**代码清单2-21使用-XX:+UseParallelGC参数**

**Heap**

**PSYoungGen total 4800K, used 1380K [0x1dac0000, 0x1e010000, 0x23010000)**

**eden space 4160K, 33% used [0x1dac0000,0x1dc19130,0x1ded0000)**

**from space 640K, 0% used [0x1df70000,0x1df70000,0x1e010000)**

**to space 640K, 0% used [0x1ded0000,0x1ded0000,0x1df70000)**

**PSOldGen total 10944K, used 0K [0x13010000, 0x13ac0000, 0x1dac0000)**

**object space 10944K, 0% used [0x13010000,0x13010000,0x13ac0000)**

**PSPermGen total 12288K, used 2056K [0x0f010000, 0x0fc10000, 0x13010000)**

**object space 12288K, 16% used [0x0f010000,0x0f2121d0,0x0fc10000)**

**代码清单2-22老年代串行回收器输出**

**[Full GC [Tenured: 1712K->1699K(10240K), 0.0071963 secs] 1712K->1699K(19456K), [Perm : 377K->372K(12288K)], 0.0072393 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]**

**代码清单2-23使用-XX:+UseParNewGC参数**

**[GC [ParNew: 825K->161K(4928K), 0.0155258 secs][Tenured: 8704K->661K(10944K), 0.0071964 secs] 9017K->661K(15872K), [Perm : 2049K->2049K(12288K)], 0.0228090 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]**

**Heap**

**par new generation total 4992K, used 179K [0x0f010000, 0x0f570000, 0x14560000)**

**eden space 4480K, 4% used [0x0f010000, 0x0f03cda8, 0x0f470000)**

**from space 512K, 0% used [0x0f470000, 0x0f470000, 0x0f4f0000)**

**to space 512K, 0% used [0x0f4f0000, 0x0f4f0000, 0x0f570000)**

**tenured generation total 10944K, used 8853K [0x14560000, 0x15010000, 0x1f010000)**

**the space 10944K, 80% used [0x14560000, 0x14e057c0, 0x14e05800, 0x15010000)**

**compacting perm gen total 12288K, used 2060K [0x1f010000, 0x1fc10000, 0x23010000)**

**the space 12288K, 16% used [0x1f010000, 0x1f213228, 0x1f213400, 0x1fc10000)**

**No shared spaces configured.**

**代码清单2-24 24个线程的Parallel收集器**

**Heap**

**PSYoungGen total 4800K, used 893K [0x1dac0000, 0x1e010000, 0x23010000)**

**eden space 4160K, 21% used [0x1dac0000,0x1db9f570,0x1ded0000)**

**from space 640K, 0% used [0x1df70000,0x1df70000,0x1e010000)**

**to space 640K, 0% used [0x1ded0000,0x1ded0000,0x1df70000)**

**ParOldGen total 19200K, used 16384K [0x13010000, 0x142d0000, 0x1dac0000)**

**object space 19200K, 85% used [0x13010000,0x14010020,0x142d0000)**

**PSPermGen total 12288K, used 2054K [0x0f010000, 0x0fc10000, 0x13010000)**

**object space 12288K, 16% used [0x0f010000,0x0f2119c0,0x0fc10000)**

**代码清单2-25年轻代并行回收收集器工作日志**

**Heap**

**PSYoungGen total 4800K, used 893K [0x1dac0000, 0x1e010000, 0x23010000)**

**eden space 4160K, 21% used [0x1dac0000,0x1db9f570,0x1ded0000)**

**from space 640K, 0% used [0x1df70000,0x1df70000,0x1e010000)**

**to space 640K, 0% used [0x1ded0000,0x1ded0000,0x1df70000)**

**PSOldGen total 19200K, used 16384K [0x13010000, 0x142d0000, 0x1dac0000)**

**object space 19200K, 85% used [0x13010000,0x14010020,0x142d0000)**

**PSPermGen total 12288K, used 2054K [0x0f010000, 0x0fc10000, 0x13010000)**

**object space 12288K, 16% used [0x0f010000,0x0f2119c0,0x0fc10000)**

**代码清单2-26 老年代并行回收收集器线程100时日志**

**Heap**

**PSYoungGen total 4800K, used 893K [0x1dac0000, 0x1e010000, 0x23010000)**

**eden space 4160K, 21% used [0x1dac0000,0x1db9f570,0x1ded0000)**

**from space 640K, 0% used [0x1df70000,0x1df70000,0x1e010000)**

**to space 640K, 0% used [0x1ded0000,0x1ded0000,0x1df70000)**

**ParOldGen total 19200K, used 16384K [0x13010000, 0x142d0000, 0x1dac0000)**

**object space 19200K, 85% used [0x13010000,0x14010020,0x142d0000)**

**PSPermGen total 12288K, used 2054K [0x0f010000, 0x0fc10000, 0x13010000)**

**object space 12288K, 16% used [0x0f010000,0x0f2119c0,0x0fc10000)**

**代码清单2-27 线程100个时使用CMS收集器日志输出**

**[GC [DefNew: 825K->149K(4928K), 0.0023384 secs][Tenured: 8704K->661K(10944K), 0.0587725 secs] 9017K->661K(15872K), [Perm : 374K->374K(12288K)], 0.0612037 secs] [Times: user=0.01 sys=0.02, real=0.06 secs]**

**Heap**

**def new generation total 4992K, used 179K [0x27010000, 0x27570000, 0x2c560000)**

**eden space 4480K, 4% used [0x27010000, 0x2703cda8, 0x27470000)**

**from space 512K, 0% used [0x27470000, 0x27470000, 0x274f0000)**

**to space 512K, 0% used [0x274f0000, 0x274f0000, 0x27570000)**

**tenured generation total 10944K, used 8853K [0x2c560000, 0x2d010000, 0x37010000)**

**the space 10944K, 80% used [0x2c560000, 0x2ce057c8, 0x2ce05800, 0x2d010000)**

**compacting perm gen total 12288K, used 374K [0x37010000, 0x37c10000, 0x3b010000)**

**the space 12288K, 3% used [0x37010000, 0x3706db10, 0x3706dc00, 0x37c10000)**

**ro space 10240K, 51% used [0x3b010000, 0x3b543000, 0x3b543000, 0x3ba10000)**

**rw space 12288K, 55% used [0x3ba10000, 0x3c0ae4f8, 0x3c0ae600, 0x3c610000)**

**代码清单2-28 使用-XX:+**UseConcMarkSweepGC**参数**

**[GC [ParNew: 8967K->669K(14784K), 0.0040895 secs] 8967K->669K(63936K), 0.0043255 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]**

**Heap**

**par new generation total 14784K, used 9389K [0x03f50000, 0x04f50000, 0x04f50000)**

**eden space 13184K, 66% used [0x03f50000, 0x047d3e58, 0x04c30000)**

**from space 1600K, 41% used [0x04dc0000, 0x04e67738, 0x04f50000)**

**to space 1600K, 0% used [0x04c30000, 0x04c30000, 0x04dc0000)**

**concurrent mark-sweep generation total 49152K, used 0K [0x04f50000, 0x07f50000, 0x09f50000)**

**concurrent-mark-sweep perm gen total 12288K, used 2060K [0x09f50000, 0x0ab50000, 0x0df50000)**

**代码清单2-29 并发收集器的线程设置8个时日志输出**

**[GC [ParNew: 8967K->676K(14784K), 0.0036983 secs] 8967K->676K(63936K), 0.0037662 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]**

**Heap**

**par new generation total 14784K, used 9395K [0x040e0000, 0x050e0000, 0x050e0000)**

**eden space 13184K, 66% used [0x040e0000, 0x04963e58, 0x04dc0000)**

**from space 1600K, 42% used [0x04f50000, 0x04ff9100, 0x050e0000)**

**to space 1600K, 0% used [0x04dc0000, 0x04dc0000, 0x04f50000)**

**concurrent mark-sweep generation total 49152K, used 0K [0x050e0000, 0x080e0000, 0x0a0e0000)**

**concurrent-mark-sweep perm gen total 12288K, used 2060K [0x0a0e0000, 0x0ace0000, 0x0e0e0000)**

**代码清单2-30 实验程序**

**import com.thoughtworks.xstream.XStream;**

**public class XStreamTest {**

**public static void main(String[] args) throws Exception {**

**while(true){**

**XStream xs = new XStream();**

**xs.toString();**

**xs = null;**

**}**

**}**

**}**

**代码清单2-31 实验程序**

**public class TestSlowYGC {**

**public static void main(String[] args) throws Exception {**

**int i= 0;**

**while (true) {**

**XStream xs = new XStream();**

**xs.toString();**

**xs = null;**

**if(i++ % 10000 == 0)**

**{**

**System.gc();**

**}**

**}**

**}**

**}**

**代码清单3-1范例程序**

**import Java.util.concurrent.\*;**

**import Java.util.\*;**

**public class GreenhouseScheduler {**

**private volatile boolean light = false;**

**private volatile boolean water = false;**

**private String thermostat = "Day";**

**public synchronized String getThermostat() {**

**return thermostat;**

**}**

**public synchronized void setThermostat(String value) {**

**thermostat = value;**

**}**

**ScheduledThreadPoolExecutor scheduler = new ScheduledThreadPoolExecutor(10);**

**public void schedule(Runnable event, long delay) {**

**scheduler.schedule(event,delay,TimeUnit.MILLISECONDS);**

**}**

**public void repeat(Runnable event, long initialDelay, long period) {**

**scheduler.scheduleAtFixedRate(**

**event, initialDelay, period, TimeUnit.MILLISECONDS);**

**}**

**class LightOn implements Runnable {**

**public void run() {**

**// Put hardware control code here to**

**// physically turn on the light.**

**System.out.println("Turning on lights");**

**light = true;**

**}**

**}**

**class LightOff implements Runnable {**

**public void run() {**

**// Put hardware control code here to**

**// physically turn off the light.**

**System.out.println("Turning off lights");**

**light = false;**

**}**

**}**

**class WaterOn implements Runnable {**

**public void run() {**

**// Put hardware control code here.**

**System.out.println("Turning greenhouse water on");**

**water = true;**

**}**

**}**

**class WaterOff implements Runnable {**

**public void run() {**

**// Put hardware control code here.**

**System.out.println("Turning greenhouse water off");**

**water = false;**

**}**

**}**

**class ThermostatNight implements Runnable {**

**public void run() {**

**// Put hardware control code here.**

**System.out.println("Thermostat to night setting");**

**setThermostat("Night");**

**}**

**}**

**class ThermostatDay implements Runnable {**

**public void run() {**

**// Put hardware control code here.**

**System.out.println("Thermostat to day setting");**

**setThermostat("Day");**

**}**

**}**

**class Bell implements Runnable {**

**public void run() { System.out.println("Bing!"); }**

**}**

**class Terminate implements Runnable {**

**public void run() {**

**System.out.println("Terminating");**

**scheduler.shutdownNow();**

**// Must start a separate task to do this job,**

**// since the scheduler has been shut down:**

**new Thread() {**

**public void run() {**

**for(DataPoint d : data)**

**System.out.println(d);**

**}**

**}.start();**

**}**

**}**

**// New feature: data collection**

**static class DataPoint {**

**final Calendar time;**

**final float temperature;**

**final float humidity;**

**public DataPoint(Calendar d, float temp, float hum) {**

**time = d;**

**temperature = temp;**

**humidity = hum;**

**}**

**public String toString() {**

**return time.getTime() +**

**String.format(**

**" temperature: %1$.1f humidity: %2$.2f",**

**temperature, humidity);**

**}**

**}**

**private Calendar lastTime = Calendar.getInstance();**

**{ // Adjust date to the half hour**

**lastTime.set(Calendar.MINUTE, 30);**

**lastTime.set(Calendar.SECOND, 00);**

**}**

**private float lastTemp = 65.0f;**

**private int tempDirection = +1;**

**private float lastHumidity = 50.0f;**

**private int humidityDirection = +1;**

**private Random rand = new Random(47);**

**List<DataPoint> data = Collections.synchronizedList(new ArrayList<DataPoint>());**

**class CollectData implements Runnable {**

**public void run() {**

**System.out.println("Collecting data");**

**synchronized(GreenhouseScheduler.this) {**

**// Pretend the interval is longer than it is:**

**lastTime.set(Calendar.MINUTE,lastTime.get(Calendar.MINUTE) + 30);**

**// One in 5 chances of reversing the direction:**

**if(rand.nextInt(5) == 4)**

**tempDirection = -tempDirection;**

**// Store previous value:**

**lastTemp = lastTemp + tempDirection \* (1.0f + rand.nextFloat());**

**if(rand.nextInt(5) == 4)**

**humidityDirection = -humidityDirection;**

**lastHumidity = lastHumidity + humidityDirection \* rand.nextFloat();**

**// Calendar must be cloned, otherwise all**

**// DataPoints hold references to the same lastTime.**

**// For a basic object like Calendar, clone() is OK.**

**data.add(new DataPoint((Calendar)lastTime.clone(),lastTemp, lastHumidity));**

**}**

**}**

**}**

**public static void main(String[] args) {**

**GreenhouseScheduler gh = new GreenhouseScheduler();**

**gh.schedule(gh.new Terminate(), 5000);**

**// Former "Restart" class not necessary:**

**gh.repeat(gh.new Bell(), 0, 1000);**

**gh.repeat(gh.new ThermostatNight(), 0, 2000);**

**gh.repeat(gh.new LightOn(), 0, 200);**

**gh.repeat(gh.new LightOff(), 0, 400);**

**gh.repeat(gh.new WaterOn(), 0, 600);**

**gh.repeat(gh.new WaterOff(), 0, 800);**

**gh.repeat(gh.new ThermostatDay(), 0, 1400);**

**gh.repeat(gh.new CollectData(), 500, 500);**

**}**

**}**

**代码清单3-2范例程序3-1运行输出**

**Bing!**

**Thermostat to night setting**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water on**

**Turning greenhouse water off**

**Thermostat to day setting**

**Turning on lights**

**Turning on lights**

**Turning off lights**

**Collecting data**

**Turning on lights**

**Turning greenhouse water on**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water off**

**Bing!**

**Turning on lights**

**Collecting data**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water on**

**Turning on lights**

**Thermostat to day setting**

**Collecting data**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water off**

**Turning on lights**

**Turning greenhouse water on**

**Bing!**

**Thermostat to night setting**

**Turning on lights**

**Turning off lights**

**Collecting data**

**Turning on lights**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water on**

**Turning greenhouse water off**

**Collecting data**

**Turning on lights**

**Turning on lights**

**Turning off lights**

**Thermostat to day setting**

**Bing!**

**Turning on lights**

**Turning greenhouse water on**

**Collecting data**

**Turning off lights**

**Turning on lights**

**Turning greenhouse water off**

**Turning on lights**

**Collecting data**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water on**

**Turning on lights**

**Bing!**

**Thermostat to night setting**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water off**

**Collecting data**

**Turning greenhouse water on**

**Thermostat to day setting**

**Turning on lights**

**Turning on lights**

**Turning off lights**

**Collecting data**

**Turning on lights**

**Turning on lights**

**Turning off lights**

**Turning greenhouse water on**

**Turning greenhouse water off**

**Terminating**

**Mon Aug 01 23:00:00 CST 2016 temperature: 66.4 humidity: 50.05**

**Mon Aug 01 23:30:00 CST 2016 temperature: 68.0 humidity: 50.47**

**Tue Aug 02 00:00:00 CST 2016 temperature: 69.7 humidity: 51.42**

**Tue Aug 02 00:30:00 CST 2016 temperature: 70.8 humidity: 50.87**

**Tue Aug 02 01:00:00 CST 2016 temperature: 72.0 humidity: 50.32**

**Tue Aug 02 01:30:00 CST 2016 temperature: 73.2 humidity: 49.92**

**Tue Aug 02 02:00:00 CST 2016 temperature: 71.9 humidity: 49.81**

**Tue Aug 02 02:30:00 CST 2016 temperature: 70.1 humidity: 50.25**

**Tue Aug 02 03:00:00 CST 2016 temperature: 68.9 humidity: 51.00代码清单3-3 Java –X运行输出**

**-Xmixed 混合模式执行 (默认)**

**-Xint 仅解释模式执行**

**-Xbootclasspath:<用 ; 分隔的目录和 zip/jar 文件>**

**设置搜索路径以引导类和资源**

**-Xbootclasspath/a:<用 ; 分隔的目录和 zip/jar 文件>**

**附加在引导类路径末尾**

**-Xbootclasspath/p:<用 ; 分隔的目录和 zip/jar 文件>**

**置于引导类路径之前**

**-Xdiag 显示附加诊断消息**

**-Xnoclassgc 禁用类垃圾收集**

**-Xincgc 启用增量垃圾收集**

**-Xloggc:<file> 将 GC 状态记录在文件中 (带时间戳)**

**-Xbatch 禁用后台编译**

**-Xms<size> 设置初始 Java 堆大小**

**-Xmx<size> 设置最大 Java 堆大小**

**-Xss<size> 设置 Java 线程堆栈大小**

**-Xprof 输出 cpu 配置文件数据**

**-Xfuture 启用最严格的检查, 预期将来的默认值**

**-Xrs 减少 Java/VM 对操作系统信号的使用 (请参阅文档)**

**-Xcheck:jni 对 JNI 函数执行其他检查**

**-Xshare:off 不尝试使用共享类数据**

**-Xshare:auto 在可能的情况下使用共享类数据 (默认)**

**-Xshare:on 要求使用共享类数据, 否则将失败。**

**-XshowSettings 显示所有设置并继续**

**-XshowSettings:all**

**显示所有设置并继续**

**-XshowSettings:vm 显示所有与 vm 相关的设置并继续**

**-XshowSettings:properties**

**显示所有属性设置并继续**

**-XshowSettings:locale**

**显示所有与区域设置相关的设置并继续**

**代码清单3-4 -XX:+PrintGCDetails运行输出**

**Heap**

**PSYoungGen total 17920K, used 5552K [0x00000000ec100000, 0x00000000ed500000, 0x0000000100000000)**

**eden space 15360K, 36% used [0x00000000ec100000,0x00000000ec66c398,0x00000000ed000000)**

**from space 2560K, 0% used [0x00000000ed280000,0x00000000ed280000,0x00000000ed500000)**

**to space 2560K, 0% used [0x00000000ed000000,0x00000000ed000000,0x00000000ed280000)**

**ParOldGen total 40960K, used 0K [0x00000000c4200000, 0x00000000c6a00000, 0x00000000ec100000)**

**object space 40960K, 0% used [0x00000000c4200000,0x00000000c4200000,0x00000000c6a00000)**

**Metaspace used 3958K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-5 -Xloggc运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1504580k free), swap 7843228k(5006548k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:-UseLargePagesIndividualAllocation -XX:+UseParallelGC**

**Heap**

**PSYoungGen total 17920K, used 5552K [0x00000000ec100000, 0x00000000ed500000, 0x0000000100000000)**

**eden space 15360K, 36% used [0x00000000ec100000,0x00000000ec66c070,0x00000000ed000000)**

**from space 2560K, 0% used [0x00000000ed280000,0x00000000ed280000,0x00000000ed500000)**

**to space 2560K, 0% used [0x00000000ed000000,0x00000000ed000000,0x00000000ed280000)**

**ParOldGen total 40960K, used 0K [0x00000000c4200000, 0x00000000c6a00000, 0x00000000ec100000)**

**object space 40960K, 0% used [0x00000000c4200000,0x00000000c4200000,0x00000000c6a00000)**

**Metaspace used 3967K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-6 -XX:+UseSerialGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1875400k free), swap 7843228k(5597040k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:-UseLargePagesIndividualAllocation -XX:+UseSerialGC**

**Heap**

**def new generation total 18432K, used 5904K [0x00000000c4200000, 0x00000000c5600000, 0x00000000d8150000)**

**eden space 16384K, 36% used [0x00000000c4200000, 0x00000000c47c43a0, 0x00000000c5200000)**

**from space 2048K, 0% used [0x00000000c5200000, 0x00000000c5200000, 0x00000000c5400000)**

**to space 2048K, 0% used [0x00000000c5400000, 0x00000000c5400000, 0x00000000c5600000)**

**tenured generation total 40960K, used 0K [0x00000000d8150000, 0x00000000da950000, 0x0000000100000000)**

**the space 40960K, 0% used [0x00000000d8150000, 0x00000000d8150000, 0x00000000d8150200, 0x00000000da950000)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-7 -XX:+UseParNewGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1857264k free), swap 7843228k(5596788k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:-UseLargePagesIndividualAllocation -XX:+UseParNewGC**

**Heap**

**par new generation total 18432K, used 5906K [0x00000000c4200000, 0x00000000c5600000, 0x00000000d8150000)**

**eden space 16384K, 36% used [0x00000000c4200000, 0x00000000c47c4800, 0x00000000c5200000)**

**from space 2048K, 0% used [0x00000000c5200000, 0x00000000c5200000, 0x00000000c5400000)**

**to space 2048K, 0% used [0x00000000c5400000, 0x00000000c5400000, 0x00000000c5600000)**

**tenured generation total 40960K, used 0K [0x00000000d8150000, 0x00000000da950000, 0x0000000100000000)**

**the space 40960K, 0% used [0x00000000d8150000, 0x00000000d8150000, 0x00000000d8150200, 0x00000000da950000)**

**Metaspace used 3950K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-8 -XX: +** **UseParNewGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM warning: Using the ParNew young collector with the Serial old collector is deprecated and will likely be removed in a future release**

**代码清单3-9 -XX:+UseParallelGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1875088k free), swap 7843228k(5614756k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:-UseLargePagesIndividualAllocation -XX:+UseParallelGC**

**Heap**

**PSYoungGen total 17920K, used 5550K [0x00000000ec100000, 0x00000000ed500000, 0x0000000100000000)**

**eden space 15360K, 36% used [0x00000000ec100000,0x00000000ec66bb08,0x00000000ed000000)**

**from space 2560K, 0% used [0x00000000ed280000,0x00000000ed280000,0x00000000ed500000)**

**to space 2560K, 0% used [0x00000000ed000000,0x00000000ed000000,0x00000000ed280000)**

**ParOldGen total 40960K, used 0K [0x00000000c4200000, 0x00000000c6a00000, 0x00000000ec100000)**

**object space 40960K, 0% used [0x00000000c4200000,0x00000000c4200000,0x00000000c6a00000)**

**Metaspace used 3954K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-10 -XX:+UseParallelOldGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1461196k free), swap 7843228k(4975780k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:-UseLargePagesIndividualAllocation -XX:+UseParallelOldGC**

**Heap**

**PSYoungGen total 17920K, used 5550K [0x00000000ec100000, 0x00000000ed500000, 0x0000000100000000)**

**eden space 15360K, 36% used [0x00000000ec100000,0x00000000ec66ba68,0x00000000ed000000)**

**from space 2560K, 0% used [0x00000000ed280000,0x00000000ed280000,0x00000000ed500000)**

**to space 2560K, 0% used [0x00000000ed000000,0x00000000ed000000,0x00000000ed280000)**

**ParOldGen total 40960K, used 0K [0x00000000c4200000, 0x00000000c6a00000, 0x00000000ec100000)**

**object space 40960K, 0% used [0x00000000c4200000,0x00000000c4200000,0x00000000c6a00000)**

**Metaspace used 3955K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-11 -XX:+UseConcMarkSweepGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1873928k free), swap 7843228k(5613224k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:MaxNewSize=174485504 -XX:MaxTenuringThreshold=6 -XX:OldPLABSize=16 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:-UseLargePagesIndividualAllocation -XX:+UseParNewGC**

**Heap**

**par new generation total 18432K, used 5904K [0x00000000c4200000, 0x00000000c5600000, 0x00000000ce860000)**

**eden space 16384K, 36% used [0x00000000c4200000, 0x00000000c47c4120, 0x00000000c5200000)**

**from space 2048K, 0% used [0x00000000c5200000, 0x00000000c5200000, 0x00000000c5400000)**

**to space 2048K, 0% used [0x00000000c5400000, 0x00000000c5400000, 0x00000000c5600000)**

**concurrent mark-sweep generation total 40960K, used 0K [0x00000000ce860000, 0x00000000d1060000, 0x0000000100000000)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-12 -XX:+UseG1GC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1585004k free), swap 7843228k(5137568k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 3 young (3072K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-13 -XX:+** **PrintGCApplicationStoppedTime运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1449048k free), swap 7843228k(4974672k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationConcurrentTime -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.159: Application time: 0.8766273 seconds**

**1.160: Total time for which application threads were stopped: 0.0001140 seconds, Stopping threads took: 0.0000351 seconds**

**4.159: Application time: 2.9998299 seconds**

**4.160: Total time for which application threads were stopped: 0.0002126 seconds, Stopping threads took: 0.0000724 seconds**

**4.418: Application time: 0.2587571 seconds**

**4.419: Total time for which application threads were stopped: 0.0001491 seconds, Stopping threads took: 0.0000381 seconds**

**5.462: Application time: 1.0433280 seconds**

**5.462: Total time for which application threads were stopped: 0.0000664 seconds, Stopping threads took: 0.0000360 seconds**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 3 young (3072K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**5.463: Application time: 0.0007787 seconds**

**代码清单3-14 -XX:ConcGCThreads运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1632868k free), swap 7843228k(5179308k free)**

**CommandLine flags: -XX:ConcGCThreads=2 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.110: Total time for which application threads were stopped: 0.0002820 seconds, Stopping threads took: 0.0000720 seconds**

**4.115: Total time for which application threads were stopped: 0.0002280 seconds, Stopping threads took: 0.0000750 seconds**

**4.346: Total time for which application threads were stopped: 0.0003227 seconds, Stopping threads took: 0.0000784 seconds**

**5.396: Total time for which application threads were stopped: 0.0000806 seconds, Stopping threads took: 0.0000416 seconds**

**Heap**

**garbage-first heap total 61440K, used 3072K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-15 -XX:G1HeapRegionSize运行输出1**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1608364k free), swap 7843228k(5151084k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=33554432 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.191: Total time for which application threads were stopped: 0.0001496 seconds, Stopping threads took: 0.0000411 seconds**

**3.191: Total time for which application threads were stopped: 0.0000694 seconds, Stopping threads took: 0.0000317 seconds**

**4.403: Total time for which application threads were stopped: 0.0001723 seconds, Stopping threads took: 0.0000386 seconds**

**5.402: Total time for which application threads were stopped: 0.0000754 seconds, Stopping threads took: 0.0000441 seconds**

**5.413: Total time for which application threads were stopped: 0.0000639 seconds, Stopping threads took: 0.0000334 seconds**

**Heap**

**garbage-first heap total 65536K, used 0K [0x00000000c4000000, 0x00000000c6000010, 0x0000000100000000)**

**region size 32768K, 1 young (32768K), 0 survivors (0K)**

**Metaspace used 3955K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-16 -XX:G1HeapRegionSize运行输出2**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1614848k free), swap 7843228k(5159776k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=1048576 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.080: Total time for which application threads were stopped: 0.0001303 seconds, Stopping threads took: 0.0000356 seconds**

**4.080: Total time for which application threads were stopped: 0.0001346 seconds, Stopping threads took: 0.0000523 seconds**

**4.374: Total time for which application threads were stopped: 0.0001740 seconds, Stopping threads took: 0.0000437 seconds**

**5.348: Total time for which application threads were stopped: 0.0000737 seconds, Stopping threads took: 0.0000360 seconds**

**Heap**

**garbage-first heap total 61440K, used 3072K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3964K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-17 -XX:G1HeapRegionSize运行输出3**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1622124k free), swap 7843228k(5165420k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.114: Total time for which application threads were stopped: 0.0001307 seconds, Stopping threads took: 0.0000347 seconds**

**4.118: Total time for which application threads were stopped: 0.0001204 seconds, Stopping threads took: 0.0000390 seconds**

**4.423: Total time for which application threads were stopped: 0.0001419 seconds, Stopping threads took: 0.0000334 seconds**

**5.450: Total time for which application threads were stopped: 0.0000759 seconds, Stopping threads took: 0.0000369 seconds**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3975K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-18 -XX:G1HeapWastePercent运行输出1**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1961388k free), swap 7843228k(5602892k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=99 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.118: Total time for which application threads were stopped: 0.0001457 seconds, Stopping threads took: 0.0000373 seconds**

**4.149: Total time for which application threads were stopped: 0.0003476 seconds, Stopping threads took: 0.0002169 seconds**

**4.305: Total time for which application threads were stopped: 0.0001590 seconds, Stopping threads took: 0.0000403 seconds**

**5.382: Total time for which application threads were stopped: 0.0000896 seconds, Stopping threads took: 0.0000381 seconds**

**Heap**

**garbage-first heap total 61440K, used 0K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 1 young (2048K), 0 survivors (0K)**

**Metaspace used 3947K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-19 -XX:G1HeapWastePercent运行输出2**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1965444k free), swap 7843228k(5606132k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=1 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.095: Total time for which application threads were stopped: 0.0002850 seconds, Stopping threads took: 0.0000729 seconds**

**4.139: Total time for which application threads were stopped: 0.0001607 seconds, Stopping threads took: 0.0000519 seconds**

**4.250: Total time for which application threads were stopped: 0.0019170 seconds, Stopping threads took: 0.0018026 seconds**

**5.347: Total time for which application threads were stopped: 0.0001196 seconds, Stopping threads took: 0.0000669 seconds**

**Heap**

**garbage-first heap total 61440K, used 0K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 1 young (2048K), 0 survivors (0K)**

**Metaspace used 3950K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-20 -XX:G1MixedGCCountTarget运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1963596k free), swap 7843228k(5602320k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=80 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.068: Total time for which application threads were stopped: 0.0001230 seconds, Stopping threads took: 0.0000321 seconds**

**4.098: Total time for which application threads were stopped: 0.0001286 seconds, Stopping threads took: 0.0000446 seconds**

**4.207: Total time for which application threads were stopped: 0.0001809 seconds, Stopping threads took: 0.0000369 seconds**

**5.317: Total time for which application threads were stopped: 0.0000716 seconds, Stopping threads took: 0.0000386 seconds**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-21 -XX:G1PrintRegionLivenessInfo运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1661092k free), swap 7843228k(5133168k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.165: Total time for which application threads were stopped: 0.0001389 seconds, Stopping threads took: 0.0000360 seconds**

**4.165: Total time for which application threads were stopped: 0.0001243 seconds, Stopping threads took: 0.0000416 seconds**

**4.483: Total time for which application threads were stopped: 0.0001551 seconds, Stopping threads took: 0.0000377 seconds**

**5.460: Total time for which application threads were stopped: 0.0006171 seconds, Stopping threads took: 0.0005734 seconds**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3950K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-22 -XX:** **G1ReservePercent运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1661008k free), swap 7843228k(5140300k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:G1ReservePercent=50 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.180: Total time for which application threads were stopped: 0.0001783 seconds, Stopping threads took: 0.0000531 seconds**

**4.180: Total time for which application threads were stopped: 0.0001046 seconds, Stopping threads took: 0.0000330 seconds**

**4.406: Total time for which application threads were stopped: 0.0001414 seconds, Stopping threads took: 0.0000356 seconds**

**5.478: Total time for which application threads were stopped: 0.0000767 seconds, Stopping threads took: 0.0000343 seconds**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3967K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-23 -XX:** **+G1SummarizeRSetStats运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1692124k free), swap 7843228k(5178744k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:G1ReservePercent=10 -XX:+G1SummarizeRSetStats -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.086: Total time for which application threads were stopped: 0.0001337 seconds, Stopping threads took: 0.0000360 seconds**

**4.086: Total time for which application threads were stopped: 0.0001037 seconds, Stopping threads took: 0.0000326 seconds**

**4.342: Total time for which application threads were stopped: 0.0001534 seconds, Stopping threads took: 0.0000373 seconds**

**5.372: Total time for which application threads were stopped: 0.0000746 seconds, Stopping threads took: 0.0000296 seconds**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3965K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**Cumulative RS summary**

**Recent concurrent refinement statistics**

**Processed 0 cards**

**Of 0 completed buffers:**

**0 ( 0.0%) by concurrent RS threads.**

**0 ( 0.0%) by mutator threads.**

**Did 0 coarsenings.**

**Concurrent RS threads times (s)**

**0.00 0.00**

**Concurrent sampling threads times (s)**

**0.00**

**Current rem set statistics**

**Total per region rem sets sizes = 154K. Max = 5K.**

**10K ( 7.0%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**143K ( 93.0%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**Static structures = 7K, free\_lists = 0K.**

**0 occupied cards represented.**

**0 ( 0.0%) entries by 2 Young regions**

**0 ( 0.0%) entries by 0 Humonguous regions**

**0 ( 0.0%) entries by 28 Free regions**

**0 ( 0.0%) entries by 0 Old regions**

**Region with largest rem set = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 5K, occupied = 0K.**

**Total heap region code root sets sizes = 1K. Max = 0K.**

**0K ( 57.9%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**0K ( 42.1%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**22 code roots represented.**

**22 (100.0%) elements by 2 Young regions**

**0 ( 0.0%) elements by 0 Humonguous regions**

**0 ( 0.0%) elements by 28 Free regions**

**0 ( 0.0%) elements by 0 Old regions**

**Region with largest amount of code roots = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 0K, num\_elems = 0.**

**代码清单3-24 -XX:** **+** **G1TraceConcRefinement运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1895664k free), swap 7843228k(5633652k free)**

**CommandLine flags: -XX:ConcGCThreads=1 -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:G1ReservePercent=10 -XX:+G1SummarizeRSetStats -XX:G1SummarizeRSetStatsPeriod=10 -XX:+G1TraceConcRefinement -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.095: Total time for which application threads were stopped: 0.0001307 seconds, Stopping threads took: 0.0000351 seconds**

**4.095: Total time for which application threads were stopped: 0.0001110 seconds, Stopping threads took: 0.0000377 seconds**

**4.370: Total time for which application threads were stopped: 0.0003351 seconds, Stopping threads took: 0.0000793 seconds**

**5.347: Total time for which application threads were stopped: 0.0000716 seconds, Stopping threads took: 0.0000394 seconds**

**G1-Refine-stop**

**G1-Refine-stop**

**G1-Refine-stop**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3949K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**Cumulative RS summary**

**Recent concurrent refinement statistics**

**Processed 0 cards**

**Of 0 completed buffers:**

**0 ( 0.0%) by concurrent RS threads.**

**0 ( 0.0%) by mutator threads.**

**Did 0 coarsenings.**

**Concurrent RS threads times (s)**

**0.00 0.00**

**Concurrent sampling threads times (s)**

**0.00**

**Current rem set statistics**

**Total per region rem sets sizes = 154K. Max = 5K.**

**10K ( 7.0%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**143K ( 93.0%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**Static structures = 7K, free\_lists = 0K.**

**0 occupied cards represented.**

**0 ( 0.0%) entries by 2 Young regions**

**0 ( 0.0%) entries by 0 Humonguous regions**

**0 ( 0.0%) entries by 28 Free regions**

**0 ( 0.0%) entries by 0 Old regions**

**Region with largest rem set = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 5K, occupied = 0K.**

**Total heap region code root sets sizes = 1K. Max = 0K.**

**0K ( 56.9%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**0K ( 43.1%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**21 code roots represented.**

**21 (100.0%) elements by 2 Young regions**

**0 ( 0.0%) elements by 0 Humonguous regions**

**0 ( 0.0%) elements by 28 Free regions**

**0 ( 0.0%) elements by 0 Old regions**

**Region with largest amount of code roots = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 0K, num\_elems = 0.**

**代码清单3-25 -XX:+G1UseAdaptiveConcRefinement运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1888412k free), swap 7843228k(5628340k free)**

**CommandLine flags: -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:G1ReservePercent=10 -XX:+G1SummarizeRSetStats -XX:G1SummarizeRSetStatsPeriod=10 -XX:+G1TraceConcRefinement -XX:+G1UseAdaptiveConcRefinement -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.058: Total time for which application threads were stopped: 0.0002893 seconds, Stopping threads took: 0.0000746 seconds**

**4.058: Total time for which application threads were stopped: 0.0001037 seconds, Stopping threads took: 0.0000339 seconds**

**4.303: Total time for which application threads were stopped: 0.0001680 seconds, Stopping threads took: 0.0000420 seconds**

**5.303: Total time for which application threads were stopped: 0.0002306 seconds, Stopping threads took: 0.0001971 seconds**

**5.312: Total time for which application threads were stopped: 0.0000673 seconds, Stopping threads took: 0.0000373 seconds**

**G1-Refine-stop**

**G1-Refine-stop**

**G1-Refine-stop**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3951K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**Cumulative RS summary**

**Recent concurrent refinement statistics**

**Processed 0 cards**

**Of 0 completed buffers:**

**0 ( 0.0%) by concurrent RS threads.**

**0 ( 0.0%) by mutator threads.**

**Did 0 coarsenings.**

**Concurrent RS threads times (s)**

**0.00 0.00**

**Concurrent sampling threads times (s)**

**0.00**

**Current rem set statistics**

**Total per region rem sets sizes = 154K. Max = 5K.**

**10K ( 7.0%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**143K ( 93.0%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**Static structures = 7K, free\_lists = 0K.**

**0 occupied cards represented.**

**0 ( 0.0%) entries by 2 Young regions**

**0 ( 0.0%) entries by 0 Humonguous regions**

**0 ( 0.0%) entries by 28 Free regions**

**0 ( 0.0%) entries by 0 Old regions**

**Region with largest rem set = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 5K, occupied = 0K.**

**Total heap region code root sets sizes = 0K. Max = 0K.**

**0K ( 55.9%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**0K ( 44.1%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**20 code roots represented.**

**20 (100.0%) elements by 2 Young regions**

**0 ( 0.0%) elements by 0 Humonguous regions**

**0 ( 0.0%) elements by 28 Free regions**

**0 ( 0.0%) elements by 0 Old regions**

**Region with largest amount of code roots = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 0K, num\_elems = 0.**

**代码清单3-26 -XX:GCTimeRatio运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1884916k free), swap 7843228k(5614188k free)**

**CommandLine flags: -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:G1ReservePercent=10 -XX:+G1SummarizeRSetStats -XX:G1SummarizeRSetStatsPeriod=10 -XX:+G1TraceConcRefinement -XX:+G1UseAdaptiveConcRefinement -XX:GCTimeRatio=90 -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.160: Total time for which application threads were stopped: 0.0003446 seconds, Stopping threads took: 0.0000986 seconds**

**4.160: Total time for which application threads were stopped: 0.0002409 seconds, Stopping threads took: 0.0000896 seconds**

**4.376: Total time for which application threads were stopped: 0.0012300 seconds, Stopping threads took: 0.0000827 seconds**

**5.406: Total time for which application threads were stopped: 0.0001153 seconds, Stopping threads took: 0.0000849 seconds**

**G1-Refine-stop**

**G1-Refine-stop**

**G1-Refine-stop**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**Cumulative RS summary**

**Recent concurrent refinement statistics**

**Processed 0 cards**

**Of 0 completed buffers:**

**0 ( 0.0%) by concurrent RS threads.**

**0 ( 0.0%) by mutator threads.**

**Did 0 coarsenings.**

**Concurrent RS threads times (s)**

**0.00 0.00**

**Concurrent sampling threads times (s)**

**0.00**

**Current rem set statistics**

**Total per region rem sets sizes = 154K. Max = 5K.**

**10K ( 7.0%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**143K ( 93.0%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**Static structures = 7K, free\_lists = 0K.**

**0 occupied cards represented.**

**0 ( 0.0%) entries by 2 Young regions**

**0 ( 0.0%) entries by 0 Humonguous regions**

**0 ( 0.0%) entries by 28 Free regions**

**0 ( 0.0%) entries by 0 Old regions**

**Region with largest rem set = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 5K, occupied = 0K.**

**Total heap region code root sets sizes = 1K. Max = 0K.**

**0K ( 56.9%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**0K ( 43.1%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**21 code roots represented.**

**21 (100.0%) elements by 2 Young regions**

**0 ( 0.0%) elements by 0 Humonguous regions**

**0 ( 0.0%) elements by 28 Free regions**

**0 ( 0.0%) elements by 0 Old regions**

**Region with largest amount of code roots = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 0K, num\_elems = 0. 代码清单3-27 -XX:** **+HeapDumpBeforeFullGC运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1893480k free), swap 7843228k(5628592k free)**

**CommandLine flags: -XX:G1HeapRegionSize=2097152 -XX:G1HeapWastePercent=5 -XX:G1MixedGCCountTarget=10 -XX:+G1PrintRegionLivenessInfo -XX:G1ReservePercent=10 -XX:+G1SummarizeRSetStats -XX:G1SummarizeRSetStatsPeriod=10 -XX:+G1TraceConcRefinement -XX:+G1UseAdaptiveConcRefinement -XX:GCTimeRatio=10 -XX:+HeapDumpAfterFullGC -XX:+HeapDumpBeforeFullGC -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**1.083: Total time for which application threads were stopped: 0.0001209 seconds, Stopping threads took: 0.0000313 seconds**

**3.083: Total time for which application threads were stopped: 0.0000741 seconds, Stopping threads took: 0.0000300 seconds**

**4.383: Total time for which application threads were stopped: 0.0001603 seconds, Stopping threads took: 0.0000334 seconds**

**5.365: Total time for which application threads were stopped: 0.0000797 seconds, Stopping threads took: 0.0000330 seconds**

**G1-Refine-stop**

**G1-Refine-stop**

**G1-Refine-stop**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c44000f0, 0x0000000100000000)**

**region size 2048K, 2 young (4096K), 0 survivors (0K)**

**Metaspace used 3973K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**Cumulative RS summary**

**Recent concurrent refinement statistics**

**Processed 0 cards**

**Of 0 completed buffers:**

**0 ( 0.0%) by concurrent RS threads.**

**0 ( 0.0%) by mutator threads.**

**Did 0 coarsenings.**

**Concurrent RS threads times (s)**

**0.00 0.00**

**Concurrent sampling threads times (s)**

**0.00**

**Current rem set statistics**

**Total per region rem sets sizes = 154K. Max = 5K.**

**10K ( 7.1%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**143K ( 92.9%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**Static structures = 7K, free\_lists = 0K.**

**0 occupied cards represented.**

**0 ( 0.0%) entries by 2 Young regions**

**0 ( 0.0%) entries by 0 Humonguous regions**

**0 ( 0.0%) entries by 28 Free regions**

**0 ( 0.0%) entries by 0 Old regions**

**Region with largest rem set = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 5K, occupied = 0K.**

**Total heap region code root sets sizes = 1K. Max = 0K.**

**0K ( 61.4%) by 2 Young regions**

**0K ( 0.0%) by 0 Humonguous regions**

**0K ( 38.6%) by 28 Free regions**

**0K ( 0.0%) by 0 Old regions**

**26 code roots represented.**

**26 (100.0%) elements by 2 Young regions**

**0 ( 0.0%) elements by 0 Humonguous regions**

**0 ( 0.0%) elements by 28 Free regions**

**0 ( 0.0%) elements by 0 Old regions**

**Region with largest amount of code roots = 29:(E)[0x00000000c7c00000,0x00000000c7e00000,0x00000000c7e00000], size = 0K, num\_elems = 0.**

**代码清单3-28 -XX:** **InitiatingHeapOccupancyPercent运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1882348k free), swap 7843228k(5605408k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:InitiatingHeapOccupancyPercent=5 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**Heap**

**garbage-first heap total 61440K, used 3072K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-29 -XX:** **InitiatingHeapOccupancyPercent运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1896116k free), swap 7843228k(5633992k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation -XX:+UseStringDeduplication**

**Heap**

**garbage-first heap total 61440K, used 3072K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3953K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-30 -XX:StringDeduplicationAgeThreshold运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1574264k free), swap 7843228k(5222608k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:StringDeduplicationAgeThreshold=1 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation -XX:+UseStringDeduplication**

**Heap**

**garbage-first heap total 61440K, used 2048K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 3 young (3072K), 0 survivors (0K)**

**Metaspace used 3954K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-31 -XX:** **+PrintStringDeduplicationStatistics运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1551852k free), swap 7843228k(5212456k free)**

**CommandLine flags: -XX:InitialHeapSize=62760512 -XX:MaxHeapSize=1004168192 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintStringDeduplicationStatistics -XX:StringDeduplicationAgeThreshold=1 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation -XX:+UseStringDeduplication**

**Heap**

**garbage-first heap total 61440K, used 3072K [0x00000000c4200000, 0x00000000c43001e0, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3948K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-32 -XX:MaxGCPauseMills运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1405972k free), swap 7843228k(4985964k free)**

**CommandLine flags: -XX:InitialHeapSize=209715200 -XX:MaxGCPauseMillis=10 -XX:MaxHeapSize=1073741824 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**Heap**

**garbage-first heap total 204800K, used 3072K [0x00000000c0000000, 0x00000000c0100640, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3968K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-33 -XX:** **MinHeapFreeRatio运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1862044k free), swap 7843228k(5638644k free)**

**CommandLine flags: -XX:InitialHeapSize=209715200 -XX:MaxHeapSize=1073741824 -XX:MinHeapFreeRatio=70 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**Heap**

**garbage-first heap total 204800K, used 3072K [0x00000000c0000000, 0x00000000c0100640, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3961K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-34 -XX:** **+PrintAdaptiveSizePolicy运行输出**

**Java HotSpot(TM) 64-Bit Server VM (25.101-b13) for windows-amd64 JRE (1.8.0\_101-b13), built on Jun 22 2016 01:21:29 by "Java\_re" with MS VC++ 10.0 (VS2010)**

**Memory: 4k page, physical 3922532k(1624280k free), swap 7843228k(5310472k free)**

**CommandLine flags: -XX:InitialHeapSize=209715200 -XX:MaxHeapFreeRatio=99 -XX:MaxHeapSize=1073741824 -XX:MinHeapFreeRatio=70 -XX:+PrintAdaptiveSizePolicy -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseG1GC -XX:-UseLargePagesIndividualAllocation**

**0.021: [G1Ergonomics (Heap Sizing) expand the heap, requested expansion amount: 209715200 bytes, attempted expansion amount: 209715200 bytes]**

**Heap**

**garbage-first heap total 204800K, used 3072K [0x00000000c0000000, 0x00000000c0100640, 0x0000000100000000)**

**region size 1024K, 4 young (4096K), 0 survivors (0K)**

**Metaspace used 3954K, capacity 4716K, committed 4864K, reserved 1056768K**

**class space used 455K, capacity 468K, committed 512K, reserved 1048576K**

**代码清单3-35 TLAB使用示例**

**public class TestUseTLAB {**

**public static void allocdemo(){**

**byte[] by = new byte[2];**

**by[0] = 1;**

**}**

**public static void main(String[] args){**

**long startTime = System.currentTimeMillis();**

**for(int i = 0;i<10000000;i++){**

**allocdemo();**

**}**

**long endTime = System.currentTimeMillis();**

**System.out.println(endTime-startTime);**

**}**

}

**代码清单3-36 诊断选项设置列表输出**

**[Global flags]**

**uintx AdaptiveSizeDecrementScaleFactor = 4 {product}**

**uintx AdaptiveSizeMajorGCDecayTimeScale = 10 {product}**

**uintx AdaptiveSizePausePolicy = 0 {product}**

**uintx AdaptiveSizePolicyCollectionCostMargin = 50 {product}**

**uintx AdaptiveSizePolicyInitializingSteps = 20**

**bool CrashOnOutOfMemoryError = false**

intx hashCode = 5 {product}

**代码清单4-1 年轻代回收示例**

**15.002:[GC pause (G1 Evacuation Pause) (young),0.014821 secs]**

**[Parallel Time: 9.5ms, GC Workers:8]**

**<snip>**

**[Code Root Fixup: 0.1 ms]**

**[Code Root Purge: 0.0 ms]**

**[Other: 4.9ms]**

**<snip>**

**[Eden: 695.0M(695.0M)->0.0B(1038.0M) Survivors:10.0m->13.0M Heap:741.5M(1175.0M)->54.0M(1175.0M)]**

**[Time:user=0.08 sys=0.00,real=0.01 secs]**

**代码清单4-2 RSets和Clear CT日志示例**

**12.540:[GC pause（G1 Evacuation Pause）（young），0.0010622 secs]**

**[Parallel Time:0.5 ms, GC Workers:8]**

**[GC Worker Start(ms):Min:0.2,Avg:0.3,Max:0.3,Diff:0.1,Sum:2.1]**

**[Update RS(ms):Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.2]**

**[Processed Buffers:Min:1,Avg:1.1,Max:2,Diff:1,Sum:9]**

**[Scan RS(ms):Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.1]**

**[Code Root Scanning(ms):Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.0]**

**[Object Copy(ms):Min:0.0,Avg:0.0,Max:0.1,Diff:0.0,Sum:0.4]**

**[Termination(ms):Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.0]**

**[GC Worker Other(ms):Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.1]**

**[GC Worker Total(ms):Min:0.3,Avg:0.4,Max:0.4,Diff:0.1,Sum:2.9]**

**[GC Worker End(ms):Min:12664.5,Avg: 12664.5,Max: 12664.5,Diff:0.0]**

**[Code Root Fixup:0.0ms]**

**[Code Root Purge:0.0ms]**

**[Clear CT:0.1ms]**

**[Other:0.4ms]**

**[Choose CSet:0.0ms]**

**[Ref Proc:0.2ms]**

**[Ref Enq:0.0ms]**

**[Redirty Cards:0.1ms]**

**[Humongous Reclaim:0.0ms]**

**[Free CSet:0.1ms]**

**[Eden:83.1M(83.1M)->0.0B(83.1M) Survivors:1024.1K->1024.1K Heap:104.1M(104.1M)->21.0M(140.1M)**

**[Times:user=0.00 sys=0.00,real=0.01 secs]**

**代码清单4-3 to-space异常日志**

**111.912:[GC pause(G1 Evacuation Pause)(young)(to-space exhausted),0.6772123 secs]**

**<snip>**

**[Evacuation Failure:330.1ms]**

**代码清单5-1 -XX:+UseG1GC运行输出**

**100.010:[GC pause (G1 Evacuation Pause) (young),0.05213secs]**

**[Parallel Time: 50.2ms, GC Workers:8]**

**[GC Worker Start(ms): Min: 100010.3, Avg: 100010.3, Max: 100010.4,Diff:0.1]**

**[Ext Root Scanning(ms):Min:0.1,Avg:0.2,Max:0.2,Diff:0.1,Sum 1.2]**

**[Update RS(ms): Min 12.8, Avg:13.0,Max:13.2,Diff:0.4,Sum 103.6]**

**[Processed Buffers: Min:15,Avg 16.0,Max:17,Diff: 2,Sum: 128]**

**[Scan RS(ms): Min:13.4,Avg:13.6,Max:13.7,Diff:0.3,Sum:109.0]**

**[Code Root Scanning(ms) : Min:0.0,Avg:0.0,Max：0.0,Diff:0.0,Sum:0.1]**

**[Object Copy(ms): Min:25.1,Avg:25.2,Max:25.2,Diff:0.1,Sum:201.5]**

**[Termination(ms): Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.1]**

**[GC Worker Other(ms):Min:0.0,Avg:0.1,Max:0.1,Diff:0.1,Sum:0.4]**

**[GC Worker Total(ms):Min:51.9,Avg:52.0,Max:52.1,Diff:0.1,Sum:416.0]**

**[GC Worker End(ms):Min:108867.5,Avg: 108867.5,Max: 108867.6,Diff:0.1]**

**[Code Root Fixup: 0.1ms]**

**[Code Root Purge: 0.0ms]**

**[Clear CT:0.2 ms]**

**[Other: 2.0ms]**

**[Choose CSet: 0.0ms]**

**[Ref Proc: 0.1ms]**

**[Ref Enq: 0.0ms]**

**[Redirty Cards: 0.2ms]**

**[Humongous Reclaim:0.0ms]**

**[Free CSet: 1.2ms]**

**[Eden: 501.0M(501.0M)->0.0B(502.0M) Survivors:23.0M->31.0M Heap:841.2M(1024.0M)->310.4M(1024.0M)]**

**代码清单5-2 8个工作线程执行的主要的并行工作**

**[GC Worker Start(ms): Min: 100010.3, Avg: 100010.3, Max: 100010.4,Diff:0.1]**

**[Ext Root Scanning(ms):Min:0.1,Avg:0.2,Max:0.2,Diff:0.1,Sum 1.2]**

**[Update RS(ms): Min 12.8, Avg:13.0,Max:13.2,Diff:0.4,Sum 103.6]**

**[Processed Buffers: Min:15,Avg 16.0,Max:17,Diff: 2,Sum: 128]**

**[Scan RS(ms): Min:13.4,Avg:13.6,Max:13.7,Diff:0.3,Sum:109.0]**

**[Code Root Scanning(ms) : Min:0.0,Avg:0.0,Max：0.0,Diff:0.0,Sum:0.1]**

**[Object Copy(ms): Min:25.1,Avg:25.2,Max:25.2,Diff:0.1,Sum:201.5]**

**[Termination(ms): Min:0.0,Avg:0.0,Max:0.0,Diff:0.0,Sum:0.1]**

**[GC Worker Other(ms):Min:0.0,Avg:0.1,Max:0.1,Diff:0.1,Sum:0.4]**

**[GC Worker Total(ms):Min:51.9,Avg:52.0,Max:52.1,Diff:0.1,Sum:416.0]**

**[GC Worker End(ms):Min:108867.5,Avg: 108867.5,Max: 108867.6,Diff:0.1]**

**代码清单5-3 每隔GC暂停阶段输出的RSets统计信息日志**

**Before GC RS summary**

**Recent concurrent refinement statistics**

**Processed 23270 cards**

**Of 96 completed buffers:**

**96（100.0%）by concurrent RS threads**

**0（ 0.0%）by mutator threads**

**Did 0 coarsenings.**

**Concurrent RS threads times（s）**

**1.11 1.11 1.11 1.11 1.11 1.11 1.11 0.00**

**Concurrent sampling threads times（s）**

**0.96**

**Current rem set statistics**

**Total per region rem sets sizes = 4380K. Max=72K.**

**767K（17.5%） by 212 Young regions**

**29K（0.7%）by 9 Humongous regions**

**2151K（49.1%）by 648 Free regions**

**1431K（32.7%）by 155 Old regions**

**Static structures = 256K,free\_lists=OK.**

**816921 occupied cards represented.**

**13670（1.7%） entires by 212 Young regions**

**4（0.0%）entires by 9 Humongous regions**

**0（0.0%）entires by 648 Free regions**

**803212（98.3%）entires by 155 Old regions**

**Region with largest rem set = 4:(0)[0x00000006c0400000,0x00000006c0500000,0x00000006c0500000],**

**Size = 72K,occupied=190K.**

**Total heap region code root sets sizes = 40K. Max = 22K.**

**3K（8.7%）by 212 Young regions**

**0K（0.3%）by 9 Humongous regions**

**10K（24.8%）by 648 Free regions**

**27K（66.2%）by 155 Old regions**

**1035 code roots represented.**

**5（0.5%）elements by 212 Young regions**

**0（0.0%）elements by 9 Humongous regions**

**0（0.0%）elements by 648 Free regions**

**1030（99.5%）elements by 155 Old regions**

**Region with largest rem set = 4:(0)[0x00000006c0400000,0x00000006c0500000,0x00000006c0500000],**

**Size = 22K,num\_elems=0.**

**After GC RS summary**

**Recent concurrent refinement statistics**

**Processed 3782 cards**

**Of 26 completed buffers:**

**26（100.0%）by concurrent RS threads**

**0（ 0.0%）by mutator threads**

**Did 0 coarsenings.**

**Concurrent RS threads times（s）**

**0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00**

**Concurrent sampling threads times（s）**

**0.00**

**Current rem set statistics**

**Total per region rem sets sizes = 4329K. Max=73K.**

**33K（0.8%） by 10 Young regions**

**29K（0.7%）by 9 Humongous regions**

**2689K（62.1%）by 810 Free regions**

**1577K（36.4%）by 195 Old regions**

**Static structures = 256K,free\_lists=63K.**

**805071 occupied cards represented.**

**0（0.0%） entires by 10 Young regions**

**4（0.0%）entires by 9 Humongous regions**

**0（0.0%）entires by 810 Free regions**

**805067（100.0%）entires by 195 Old regions**

**Region with largest rem set = 4:(0)[0x00000006c0400000,0x00000006c0500000,0x00000006c0500000],**

**Size = 73K,occupied=190K.**

**Total heap region code root sets sizes = 40K. Max = 22K.**

**0K（0.8%）by 10 Young regions**

**0K（0.3%）by 9 Humongous regions**

**12K（30.9%）by 810 Free regions**

**27K（68.0%）by 195 Old regions**

**1036 code roots represented.**

**2（0.2%）elements by 10 Young regions**

**0（0.0%）elements by 9 Humongous regions**

**0（0.0%）elements by 810 Free regions**

**1034（99.8%）elements by 195 Old regions**

**Region with largest rem set = 4:(0)[0x00000006c0400000,0x00000006c0500000,0x00000006c0500000],**

**Size = 22K,num\_elems=0.**

**代码清单5-4 重点关注**

**Processed 23270 cards**

**Of 96 completed buffers:**

**96（100.0%）by concurrent RS threads**

**0（ 0.0%）by mutator threads**

**Did 0 coarsenings.**

**代码清单5-5 -XX:MaxGCPauseMills运行结果1**

**6.311:[GC pause(G1 Evacuation Pause)(young) 6.311:[G1Ergonomics(CSet Construction) start choosing CSet,\_pending\_cards:5800,predicated base time:20.39ms,remaining time:179.61ms,target pause time:200.00ms]**

**6.311:[G1Ergonomics(CSet Construction) add young regions to CSet,eden:225 regions, survivor:68 regions, predicated young region time:202.05ms]**

**6.311:[G1Ergonomics(CSet Construction) finish choosing CSet, eden:225 regions，survivors:68 regions, old: 0 regions, predicated pause time:222.44ms, target pause time: 200.00ms],0.1126132 secs]**

**代码清单5-6 -XX:MaxGCPauseMills运行结果2**

**6.317:[GC pause(G1 Evacuation Pause)(young) 6.317:[G1Ergonomics(CSet Construction) start choosing CSet,\_pending\_cards:9129,predicated base time:14.46ms,remaining time:35.54ms,target pause time:50.00ms]**

**6.317:[G1Ergonomics(CSet Construction) add young regions to CSet,eden:284 regions, survivor:16 regions, predicated young region time:60.90ms]**

**6.317:[G1Ergonomics(CSet Construction) finish choosing CSet, eden:284 regions，survivors:16 regions, old: 0 regions, predicated pause time:75.36ms, target pause time: 50.00ms],0.0218629 secs]**

**代码清单5-7 -XX:MaxGCPauseMills运行结果3**

**6.318:[GC pause(G1 Evacuation Pause)(young) 6.318:[G1Ergonomics(CSet Construction) start choosing CSet,\_pending\_cards:5518,predicated base time:10.00ms,remaining time:40.00ms,target pause time:50.00ms]**

**6.318:[G1Ergonomics(CSet Construction) add young regions to CSet,eden:475 regions, survivor:25 regions, predicated young region time:168.35ms]**

**6.318:[G1Ergonomics(CSet Construction) finish choosing CSet, eden:475 regions，survivors:25 regions, old: 0 regions, predicated pause time:178.35ms, target pause time: 50.00ms],0.0507471 secs]**

**代码清单5-8 回收失败日志案例**

**276.731:[GC pause (G1 Evacuation Pause) (young) (to-space exhausted)，0.82729 secs]**

**[Parallel Time: 387.0ms，GC Workers:8]**

**[GC Worker Start(ms)：Min：276731.9，Avg：276731.9，Max：276732.1，Diff：0.2]**

**[Ext Root Scanning(ms)：Min：0.0，Avg：0.2，Max：0.2，Diff：0.2，Sum：1.3]**

**[Update RS(ms)：Min：17.0，Avg：17.2，Max：17.3，Diff：0.4，Sum：137.3]**

**[Processed Buffers：Min：19，Avg：21.0，Max：23，Diff：4，Sum：168]**

**[Scan RS(ms)：Min:10.5，Avg:10.7，Max：10.9，Diff：0.4，Sum：85.4]**

**[Code Root Scanning(ms)：Min：0.0，Avg：0.0，Max：0.0，Diff：0.0，Sum：0.1]**

**[Object Copy(ms)：Min：358.7，Avg：358.8，Max：358.9，Diff：0.2，Sum：2870.3]**

**[Termination(ms)：Min：0.0，Avg：0.1，Max：0.1，Diff：0.1，Sum：0.7]**

**[GC Worker Other(ms)：Min：0.0，Avg：0.0，Max：0.0，Diff：0.0，Sum：0.2]**

**[GC Worker Total (ms)：Min：386.7，Avg：386.9，Max：387.0，Diff：0.2，Sum：3095.3]**

**[GC Worker End (ms)：Min：277118.8，Avg：277118.8，Max：277118.8，Diff：0.0]**

**[Other：440.0ms]**

**[Evacuation Failure：437.5ms]**

**[Choose CSet：0.0ms]**

**[Ref Proc：0.1ms]**

**[Ref Enq：0.0ms]**

**[Redirty Cards：0.9ms]**

**[Humongous Reclaim：0.0ms]**

**[Free CSet：0.9ms]**

**[Eden：831.0M(900M)->0.0B(900M) Survivors：0.0B->0.0B Heap：1020.1M(1024.0M)->1020.1M(1024.0M)]**

**[Time：user=3.64 sys=0.20,real=0.83 secs]**

**代码清单5-9 混合回收日志示例片段1**

**97.859:[GC pause(G1 Evacuation Pause) (mixed)97.859:[G1Ergonomics((CSet Construction)start choosing CSet,\_pending\_cards:28330,predicted base time:17.45ms,remaining time:182.55ms,target pause time:200.00ms]]**

**97.859:[G1Ergonomics(CSet Construction)ass young regions to CSet,eden:37 regions,survivors:14 regions,predicted young region time:16.12ms]**

**97.859:[G1Ergonomics(CSet Construction) finish adding old regions to CSet,reason:old CSet region num reached max,old:103 regions,max:103 regions]**

**97.859:[G1Ergonomics(CSet Construction) finish choosing CSet,eden:37 regions,survivors:14 regions,old:103 regions,predicted pause time:123.38ms,target pause time:200.00ms]**

**97.905:[G1Ergonomics(Mixed GCs)continue mixed GCs,reason:candidate old regions available,candidate old regions:160 regions,reclaimable:66336328 bytes(6.18%),threshold:5.00%],0.0467861 secs]**

**代码清单5-10混合回收日志示例片段2**

**123.563:[GC pause(G1 Evacuation Pause)(mixed) 123.563:[G1Ergonomics(CSet Construction)start choosing CSet,\_pending\_cards:7404,predicated base time:6.13ms,remaining time:43.87ms,target pause time:50.00ms]**

**123.563:[G1Ergonomics(CSet Construction) finish adding old regions to CSet,reason: predicted time is too high, predicted time:0.70ms,remaining time:0.00ms,old:24 regions,min:24 regions]**

**123.563:[G1Ergonomics(CSet Construction) added expensive regions to CSet,reason:old CSet region num not reached min,old:24 regions,expensive:24 regions,min:24 regions,remaining time:0.00ms]**

**123.563:[G1 Ergonomics(CSet Construction) finish choosing CSet,eden:464 regions,survivors:36 regions,old:24 regions, predicted pause time:101.83 ms,target pause time:50.00ms]**

**123.640:[ G1 Ergonomics(Mixed GCs) continue mixed GCs,reason:candidate old regions available,candidate old regions:165 regions,reclaimable:109942200 bytes(10.24%),threshold:5.00%,0.0771342 secs ]**

**代码清单5-11混合回收日志示例片段3**

**97.859:[GC pause(G1 Evacuation Pause)(mixed)97.859:[G1Ergonomics(CSet Construction) start choosing CSet,\_pending\_cards:28330,predicted base time:17.45ms,remaining time:182.55ms,target pause time:200.00ms]**

**97.859:[G1Ergonomics(CSet Construction) add young regions to CSet,eden:37 regions, survivor:14 regions, predicted young region time:16.12ms]**

**97.859:[G1Ergonomics(CSet Construction) finish adding old regions to CSet,reason:old CSet region num reached max, old:103 regions, max:103 regions]**

**97.859:[G1Ergonomics(CSet Construction) finish choosing CSet, eden:37 regions, survivor:14 regions,old:103 regions,predicted pause time:123.38ms,target pause time:200.00ms]**

**97.905:[G1Ergonomics(Mixed GCs) continue mixed GCs,reason:candidate old regions available,candidate old regions:160 regions, reclaimable:66336328 bytes(6.18%),threshold:5.00%]**

**代码清单5-12大对象日志示例**

**91.890:[G1Ergonomics(Concurrent Cycles) request concurrent cycle initiation,reason:occupancy higher than threshold,occupancy:483393536 bytes,allocation request:2097168 bytes,threshold:483183810 bytes(45.00%),source:concurrent humongous allocation]**

**代码清单5-13 观察引用处理**

**[Other:9.9ms]**

**[Choose CSet:0.0ms]**

**[Ref Proc:8.2ms]**

**[Ref Enq:0.3ms]**

**[Redirty Cards:0.7ms]**

**[Humongous Reclaim:0.0ms]**

**[Free CSet:0.5ms]**

**代码清单6-1 设置/etc/profile方法**

**export JAVA\_HOME=/usr/lib/jdk1.7.0\_79**

**export JRE\_HOME=$JAVA\_HOME/jre**

**export CLASSPATH=$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH**

**export PATH=$JAVA\_HOME/lib:$JRE\_HOME/bin:$PATH**

**代码清单6-2 HelloWorld源代码**

**packagecom.hikvision.ivms.server;**

**importJava.lang.reflect.Method;**

**publicclass HelloWorld {**

**privatestatic String helloWorldString = "Hello, world!";**

**privatestaticvolatileinthelloWorldTrigger = 0;**

**privatestaticfinalbooleanuseMethodInvoke = false;**

**privatestatic Object lock = new Object();**

**publicstaticvoid main(String[] args) {**

**intfoo = a();**

**System.out.println("HelloWorld exiting. a() = " + foo);**

**}**

**privatestaticint a() {**

**return 1 + b();**

**}**

**privatestaticint b() {**

**return 1 + c();**

**}**

**privatestaticint c() {**

**return 1 + d("Hi");**

**}**

**privatestaticint d(String x) {**

**System.out.println("HelloWorld.d() received \"" + x + "\" as argument");**

**synchronized(lock) {**

**if (useMethodInvoke) {**

**try {**

**Method method = HelloWorld.class.getMethod("e");**

**Integer result = (Integer) method.invoke(null, new Object[0]);**

**returnresult.intValue();**

**}**

**catch (Exception e) {**

**thrownewRuntimeException(e.toString());**

**}**

**} else {**

**inti = fib(10); // 89**

**longl = i;**

**floatf = i;**

**doubled = i;**

**charc = (char) i;**

**shorts = (short) i;**

**byteb = (byte) i;**

**intret = e();**

**System.out.println("Tenth Fibonacci number in all formats: " +**

**i + ", " +**

**l + ", " +**

**f + ", " +**

**d + ", " +**

**c + ", " +**

**s + ", " +**

**b);**

**returnret;**

**}**

**}**

**}**

**publicstaticint e() {**

**System.out.println("Going to sleep...");**

**inti = 0;**

**while (helloWorldTrigger == 0) {**

**if (++i == 1000000) {**

**System.gc();**

**}**

**}**

**System.out.println(helloWorldString);**

**while (helloWorldTrigger != 0) {**

**}**

**returni;**

**}**

**// Tree-recursive implementation for test**

**publicstaticint fib(intn) {**

**if (n< 2) {**

**return 1;**

**}**

**returnfib(n - 1) + fib(n - 2);**

**}**

**}**

**代码清单6-3 HSDB绑定进程异常提示**

**C:\Users\Administrator>java -cp .;%JAVA\_HOME%/lib/sa-jdi.jar sun.jvm.hotspot.HSDB**

**Exception in thread "Thread-1" java.lang.UnsatisfiedLinkError: Can't load librar**

**y: C:\Java\jre1.8.0\_101\bin\sawindbg.dll**

**at java.lang.ClassLoader.loadLibrary(Unknown Source)**

**at java.lang.Runtime.load0(Unknown Source)**

**at java.lang.System.load(Unknown Source)**

**at sun.jvm.hotspot.debugger.windbg.WindbgDebuggerLocal.<clinit>(WindbgDe**

**buggerLocal.java:651)**

**at sun.jvm.hotspot.HotSpotAgent.setupDebuggerWin32(HotSpotAgent.java:521**

**)**

**at sun.jvm.hotspot.HotSpotAgent.setupDebugger(HotSpotAgent.java:336)**

**at sun.jvm.hotspot.HotSpotAgent.go(HotSpotAgent.java:313)**

**at sun.jvm.hotspot.HotSpotAgent.attach(HotSpotAgent.java:157)**

**at sun.jvm.hotspot.HSDB.attach(HSDB.java:1168)**

**at sun.jvm.hotspot.HSDB.access$1700(HSDB.java:53)**

**at sun.jvm.hotspot.HSDB$25$1.run(HSDB.java:436)**

**at sun.jvm.hotspot.utilities.WorkerThread$MainLoop.run(WorkerThread.java**

**:66)**

**at java.lang.Thread.run(Unknown Source)**

**代码清单6-4 生成Core Dump文件（Windows）**

**C:\Users\zhoumingyao>%JAVA\_HOME%/bin/jmap-dump:live,format=b,file=c:/heamdump.out 6096**

**Dumping heap to C:\heamdump.out ...**

**File created**

**代码清单6-4** universe命令运行结果

**hsdb> universe**

**Heap Parameters:**

**Gen 0: eden [0x00000000fa400000,0x00000000fa4aad68,0x00000000fa6b0000) space capacity = 2818048, 24.831088753633722 used**

**from [0x00000000fa6b0000,0x00000000fa6b0000,0x00000000fa700000) space capacity = 327680, 0.0 used**

**to [0x00000000fa700000,0x00000000fa700000,0x00000000fa750000) space capacity = 327680, 0.0 usedInvocations: 0**

**Gen 1: old [0x00000000fa750000,0x00000000fa750000,0x00000000fae00000) space capacity = 7012352, 0.0 usedInvocations: 0**

**perm [0x00000000fae00000,0x00000000fb078898,0x00000000fc2c0000) space capacity = 21757952, 11.90770160721009 usedInvocations: 0**