第05章：分析GC日志

第05章：分析GC日志 1

1 01-GC日志参数 4

1.1 -verbose:gc 4

1.1.1 输出gc日志信息，默认输出到标准输出 4

1.2 -XX:+PrintGC 4

1.2.1 输出GC日志。类似：-verbose:gc 4

1.3 -XX:+PrintGCDetails 4

1.3.1 在发生垃圾回收时打印内存回收相处的日志， 并在进程退出时输出当前内存各区域分配情况 4

1.4 -XX:+PrintGCTimeStamps 4

1.4.1 输出GC发生时的时间戳 4

1.5 -XX:+PrintGCDateStamps 4

1.5.1 输出GC发生时的时间戳（以日期的形式，例如：2013-05-04T21:53:59.234+0800） 4

1.6 -XX:+PrintHeapAtGC 4

1.6.1 每一次GC前和GC后，都打印堆信息 4

1.7 -Xloggc:<file> 4

1.7.1 表示把GC日志写入到一个文件中去，而不是打印到标准输出中 5

2 02-GC日志格式 5

2.1 复习：GC分类 5

2.2 不同GC分类的GC细节 6

2.2.1 老年代使用CMS GC 6

2.2.2 新生代使用Serial GC 7

2.3 GC日志分类 8

2.3.1 MinorGC 8

2.3.2 FullGC 9

2.4 GC日志结构剖析 9

2.4.1 垃圾收集器 9

2.4.2 GC前后情况 10

2.4.3 GC时间 10

2.5 Minor GC 日志解析 10

2.5.1 2020-11-20T17:19:43.265-0800 10

日志打印时间 日期格式 如 2013-05-04T21:53:59.234+0800 11

2.5.2 0.822: 11

gc发生时，Java虚拟机启动以来经过的秒数 12

2.5.3 [GC(Allocation Failure) 12

发生了一次垃圾回收，这是一次Minior GC。它不区分新生代还是老年代GC，括号里的内容是gc发生的原因，这里的Allocation Failure的原因是新生代中没有足够区域能够存放需要分配的数据而失败 12

2.5.4 [PSYoungGen:76800K->8433K(89600K) 12

PSYoungGen：表示GC发生的区域，区域名称与使用的GC收集器是密切相关的 12

Serial收集器：Default New Generation 显示Defnew 12

ParNew收集器：ParNew 12

Parallel Scanvenge收集器：PSYoung 12

老年代和新生代同理，也是和收集器名称相关 12

76800K->8433K(89600K)：GC前该内存区域已使用容量->GC后盖区域容量(该区域总容量) 12

如果是新生代，总容量则会显示整个新生代内存的9/10，即eden+from/to区 12

如果是老年代，总容量则是全身内存大小，无变化 12

2.5.5 76800K->8449K(294400K) 13

在显示完区域容量GC的情况之后，会接着显示整个堆内存区域的GC情况：GC前堆内存已使用容量->GC后堆内存容量（堆内存总容量），并且堆内存总容量 = 9/10 新生代 + 老年代，然后堆内存总容量肯定小于初始化的内存大小 13

2.5.6 ,0.0088371 13

整个GC所花费的时间，单位是秒 13

2.5.7 [Times：user=0.02 sys=0.01,real=0.01 secs] 13

user：指CPU工作在用户态所花费的时间 13

sys：指CPU工作在内核态所花费的时间 13

real：指在此次事件中所花费的总时间 13

2.6 Full GC 日志解析 13

2.6.1 2020-11-20T17:19:43.794-0800 13

日志打印时间 日期格式 如 2013-05-04T21:53:59.234+0800 14

2.6.2 1.351 14

gc发生时，Java虚拟机启动以来经过的秒数 15

2.6.3 Full GC(Metadata GCThreshold) 15

括号中是gc发生的原因，原因：Metaspace区不够用了。 除此之外，还有另外两种情况会引起Full GC，如下： 1、Full GC(FErgonomics) 原因：JVM自适应调整导致的GC 2、Full GC（System） 原因：调用了System.gc()方法 15

2.6.4 [PSYoungGen: 100082K->0K(89600K)] 15

PSYoungGen：表示GC发生的区域，区域名称与使用的GC收集器是密切相关的 15

Serial收集器：Default New Generation 显示DefNew 15

ParNew收集器：ParNew 15

Parallel Scanvenge收集器：PSYoungGen 15

老年代和新生代同理，也是和收集器名称相关 15

10082K->0K(89600K)：GC前该内存区域已使用容量->GC该区域容量(该区域总容量) 15

如果是新生代，总容量会显示整个新生代内存的9/10，即eden+from/to区 15

如果是老年代，总容量则是全部内存大小，无变化 16

2.6.5 ParOldGen：32K->9638K(204800K) 16

老年代区域没有发生GC，因此本次GC是metaspace引起的 16

2.6.6 10114K->9638K(294400K), 16

在显示完区域容量GC的情况之后，会接着显示整个堆内存区域的GC情况：GC前堆内存已使用容量->GC后堆内存容量（堆内存总容量），并且堆内存总容量 = 9/10 新生代 + 老年代，然后堆内存总容量肯定小于初始化的内存大小 16

2.6.7 [Meatspace:20158K->20156K(1067008K)], 16

metaspace GC 回收2K空间 16

3 03-GC日志分析工具 16

3.1 GCEasy 17

3.1.1 基本概述 17

3.2 GCViewer 17

3.2.1 基本概述 17

3.2.2 安装 17

3.3 其他工具 18

3.3.1 GChisto 18

官网上没有下载的地方，需要自己从SVN上拉下来编译 18

不过这个工具似乎没怎么维护了，存在不少bug 18

3.3.2 HPjmeter 18

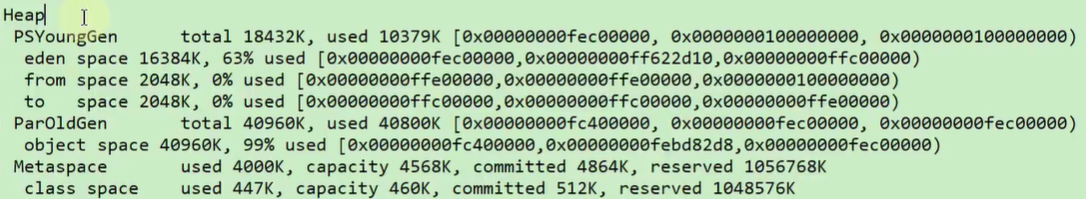
工具很强大，但是只能打开由以下参数生成的GC log，-verbose:gc -Xloggc:gc.log。添加其他参数生成的gc.log无法打开 18

HPjmeter集成了以前的HPjtune功能，可以分析在HP机器上产生的垃圾回收日志文件 18

1. 01-GC日志参数
   1. -verbose:gc
      1. 输出gc日志信息，默认输出到标准输出
   2. -XX:+PrintGC
      1. 输出GC日志。类似：-verbose:gc
   3. -XX:+PrintGCDetails
      1. 在发生垃圾回收时打印内存回收相处的日志， 并在进程退出时输出当前内存各区域分配情况
   4. -XX:+PrintGCTimeStamps
      1. 输出GC发生时的时间戳
   5. -XX:+PrintGCDateStamps
      1. 输出GC发生时的时间戳（以日期的形式，例如：2013-05-04T21:53:59.234+0800）
   6. -XX:+PrintHeapAtGC
      1. 每一次GC前和GC后，都打印堆信息
   7. -Xloggc:<file>
      1. 表示把GC日志写入到一个文件中去，而不是打印到标准输出中
2. 02-GC日志格式
   1. 复习：GC分类



1. **新生代收集：**当Eden区满的时候就会进行新生代收集，所以新生代收集和S0区域和S1区域无关
2. **老年代收集和新生代收集的关系：**进行老年代收集之前会先进行一次年轻代的垃圾收集，原因如下：一个比较大的对象无法放入新生代，那它自然会往老年代去放，如果老年代也放不下，那会先进行一次新生代的垃圾收集，之后尝试往新生代放，如果还是放不下，才会进行老年代的垃圾收集，之后在往老年代去放，这是一个过程，我来说明一下为什么需要往老年代放，但是放不下，而进行新生代垃圾收集的原因，这是因为新生代垃圾收集比老年代垃圾收集更加简单，这样做可以节省性能
3. **进行垃圾收集的时候，堆包含新生代、老年代、元空间/永久代：**可以看出Heap后面包含着新生代、老年代、元空间，但是我们设置堆空间大小的时候设置的只是新生代、老年代而已，元空间是分开设置的



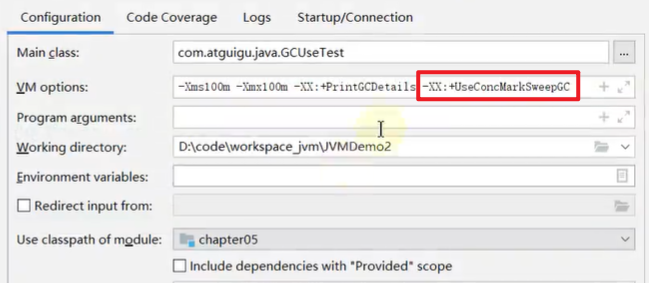
1. **哪些情况会触发Full GC：**老年代空间不足、方法区空间不足、显示调用System.gc()、Minior GC进入老年代的数据的平均大小 大于 老年代的可用内存、大对象直接进入老年代，而老年代的可用空间不足

* 1. 不同GC分类的GC细节

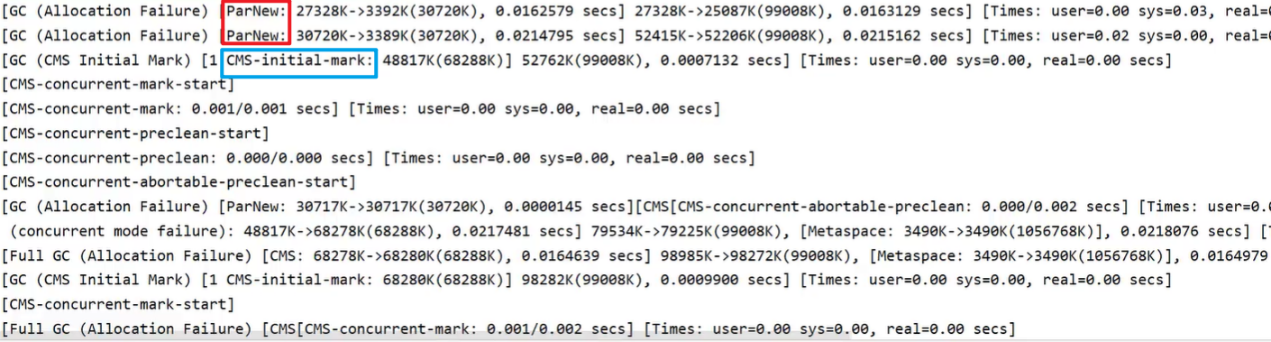
*/\*\*  
 \* -XX:+PrintCommandLineFlags  
 \*  
 \* -XX:+UseSerialGC:表明新生代使用Serial GC ，同时老年代使用Serial Old GC  
 \*  
 \* -XX:+UseParNewGC：标明新生代使用ParNew GC  
 \*  
 \* -XX:+UseParallelGC:表明新生代使用Parallel GC  
 \* -XX:+UseParallelOldGC : 表明老年代使用 Parallel Old GC  
 \* 说明：二者可以相互激活  
 \*  
 \* -XX:+UseConcMarkSweepGC：表明老年代使用CMS GC。同时，年轻代会触发对ParNew 的使用  
 \** ***@author*** *shkstart  
 \** ***@create*** *17:19  
 \*/***public class** GCUseTest {  
 **public static void** main(String[] args) {  
 ArrayList<**byte**[]> list = **new** ArrayList<>();  
  
 **while**(**true**){  
 **byte**[] arr = **new byte**[1024 \* 10];*//10kb* list.add(arr);  
*// try {  
// Thread.sleep(5);  
// } catch (InterruptedException e) {  
// e.printStackTrace();  
// }* }  
 }  
}

* + 1. 老年代使用CMS GC

**GC设置方法：**参数中使用-XX:+UseConcMarkSweepGC，说明老年代使用CMS GC，同时年轻代也会触发对ParNew的使用，因此添加该参数之后，新生代使用ParNew GC，而老年代使用CMS GC，整体是并发垃圾收集，主打低延迟

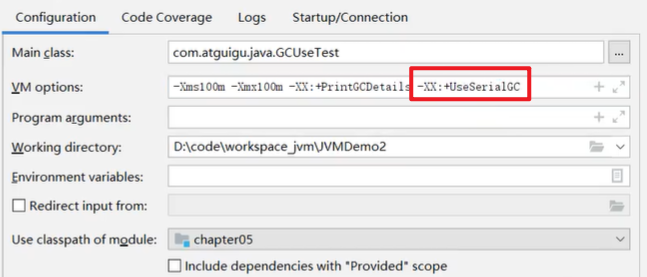


**打印出来的GC细节：**

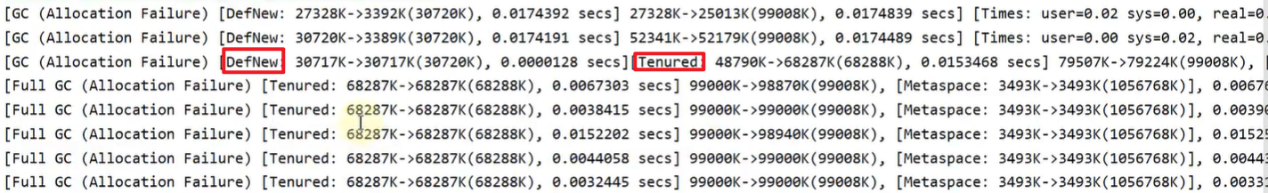


* + 1. 新生代使用Serial GC

**GC设置方法：**参数中使用-XX:+UseSerialGC，说明新生代使用Serial GC，同时老年代也会触发对Serial Old GC的使用，因此添加该参数之后，新生代使用Serial GC，而老年代使用Serial Old GC，整体是串行垃圾收集

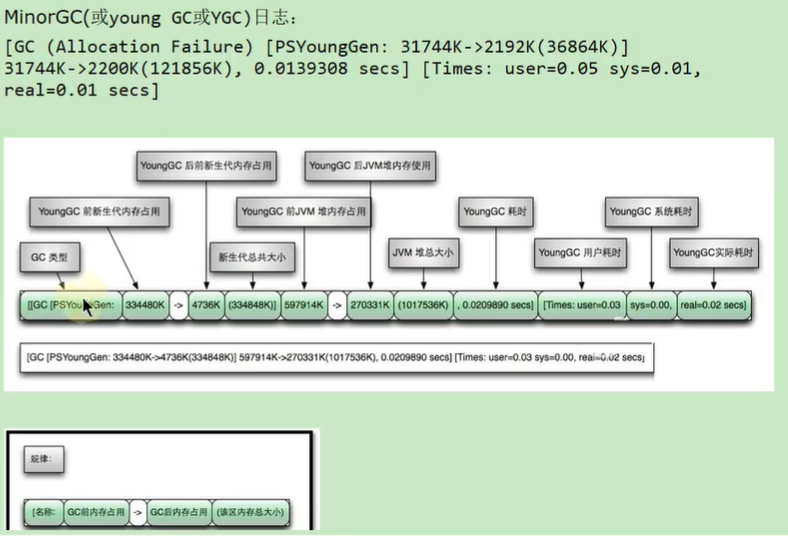


**打印出来的GC细节：**

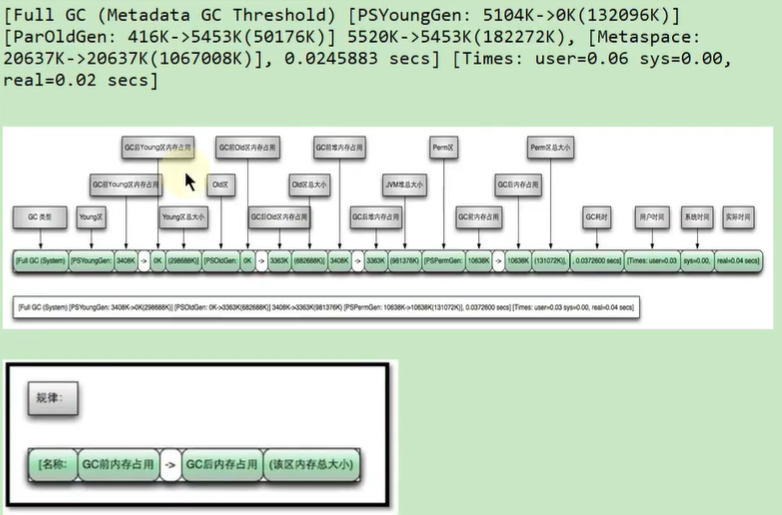


DefNew代表新生代使用Serial GC，然后Tenured代表老年代使用Serial Old GC

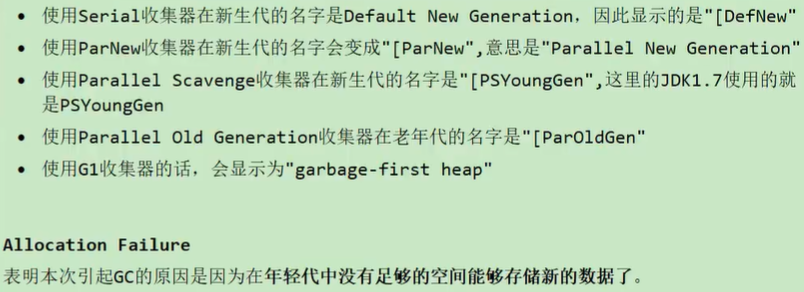
* 1. GC日志分类
     1. MinorGC



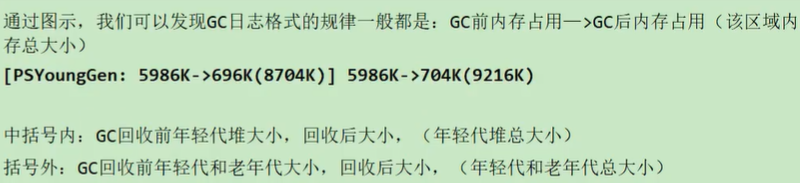
* + 1. FullGC



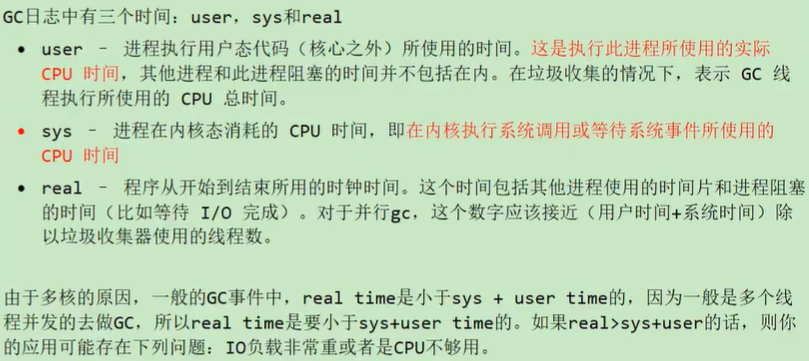
* 1. GC日志结构剖析
     1. 垃圾收集器



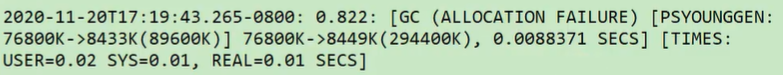
* + 1. GC前后情况



* + 1. GC时间



* 1. Minor GC 日志解析



* + 1. 2020-11-20T17:19:43.265-0800

添加-XX:+PrintGCDateStamps参数

日志打印时间 日期格式 如 2013-05-04T21:53:59.234+0800

* + 1. 0.822:

添加-XX:+PrintGCTimeStamps该参数

gc发生时，Java虚拟机启动以来经过的秒数

* + 1. [GC(Allocation Failure)

发生了一次垃圾回收，这是一次Minior GC。它不区分新生代还是老年代GC，括号里的内容是gc发生的原因，这里的Allocation Failure的原因是新生代中没有足够区域能够存放需要分配的数据而失败

* + 1. [PSYoungGen:76800K->8433K(89600K)

PSYoungGen：表示GC发生的区域，区域名称与使用的GC收集器是密切相关的

Serial收集器：Default New Generation 显示Defnew

ParNew收集器：ParNew

Parallel Scanvenge收集器：PSYoung

老年代和新生代同理，也是和收集器名称相关

76800K->8433K(89600K)：GC前该内存区域已使用容量->GC后盖区域容量(该区域总容量)

如果是新生代，总容量则会显示整个新生代内存的9/10，即eden+from/to区

如果是老年代，总容量则是全身内存大小，无变化

* + 1. 76800K->8449K(294400K)

虽然本次是Minor GC，只会进行新生代的垃圾收集，但是也肯定会打印堆中总容量相关信息

在显示完区域容量GC的情况之后，会接着显示整个堆内存区域的GC情况：GC前堆内存已使用容量->GC后堆内存容量（堆内存总容量），并且堆内存总容量 = 9/10 新生代 + 老年代，然后堆内存总容量肯定小于初始化的内存大小

* + 1. ,0.0088371

整个GC所花费的时间，单位是秒

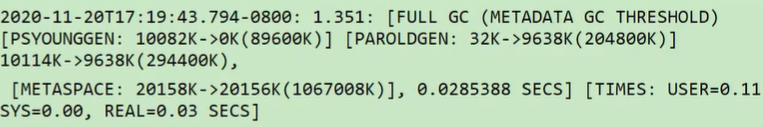
* + 1. [Times：user=0.02 sys=0.01,real=0.01 secs]

user：指CPU工作在用户态所花费的时间

sys：指CPU工作在内核态所花费的时间

real：指在此次事件中所花费的总时间

* 1. Full GC 日志解析



* + 1. 2020-11-20T17:19:43.794-0800

添加-XX:+PrintGCDateStamps参数

日志打印时间 日期格式 如 2013-05-04T21:53:59.234+0800

* + 1. 1.351

添加-XX:+PrintGCTimeStamps该参数

gc发生时，Java虚拟机启动以来经过的秒数

* + 1. Full GC(Metadata GCThreshold)

括号中是gc发生的原因，原因：Metaspace区不够用了。 除此之外，还有另外两种情况会引起Full GC，如下： 1、Full GC(FErgonomics) 原因：JVM自适应调整导致的GC 2、Full GC（System） 原因：调用了System.gc()方法

* + 1. [PSYoungGen: 100082K->0K(89600K)]

PSYoungGen：表示GC发生的区域，区域名称与使用的GC收集器是密切相关的

Serial收集器：Default New Generation 显示DefNew

ParNew收集器：ParNew

Parallel Scanvenge收集器：PSYoungGen

老年代和新生代同理，也是和收集器名称相关

10082K->0K(89600K)：GC前该内存区域已使用容量->GC该区域容量(该区域总容量)

如果是新生代，总容量会显示整个新生代内存的9/10，即eden+from/to区

如果是老年代，总容量则是全部内存大小，无变化

* + 1. ParOldGen：32K->9638K(204800K)

老年代区域没有发生GC，因此本次GC是metaspace引起的

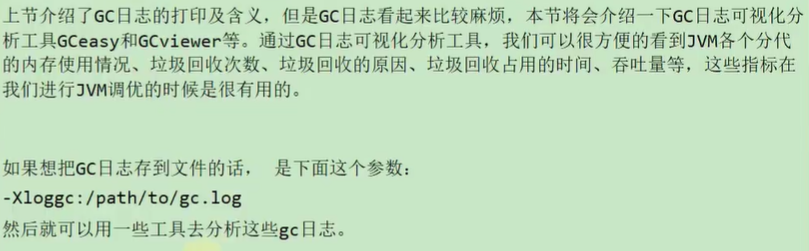
* + 1. 10114K->9638K(294400K),

在显示完区域容量GC的情况之后，会接着显示整个堆内存区域的GC情况：GC前堆内存已使用容量->GC后堆内存容量（堆内存总容量），并且堆内存总容量 = 9/10 新生代 + 老年代，然后堆内存总容量肯定小于初始化的内存大小

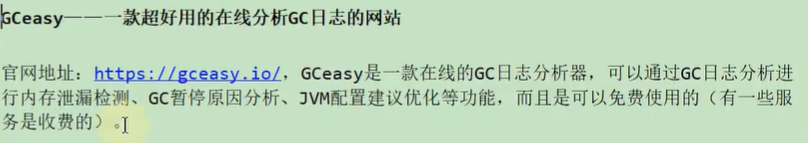
* + 1. [Meatspace:20158K->20156K(1067008K)],

metaspace GC 回收2K空间

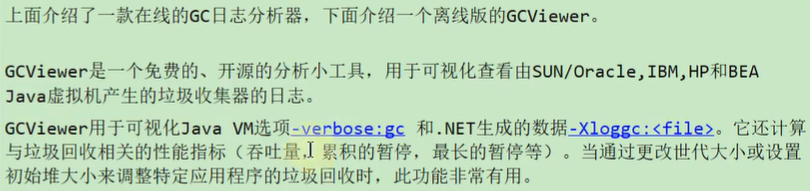
1. 03-GC日志分析工具



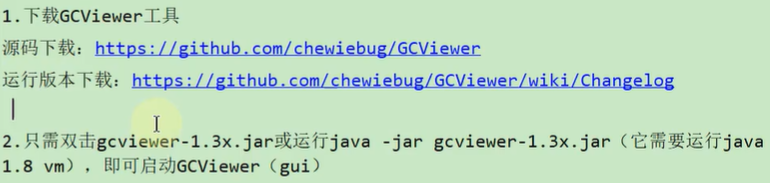
* 1. GCEasy
     1. 基本概述



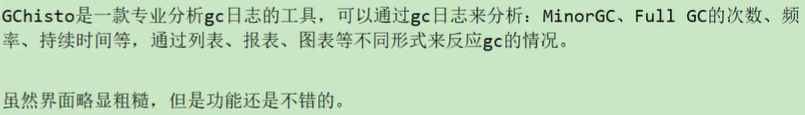
* 1. GCViewer
     1. 基本概述



* + 1. 安装



* 1. 其他工具
     1. GChisto



官网上没有下载的地方，需要自己从SVN上拉下来编译

不过这个工具似乎没怎么维护了，存在不少bug

* + 1. HPjmeter

工具很强大，但是只能打开由以下参数生成的GC log，-verbose:gc -Xloggc:gc.log。添加其他参数生成的gc.log无法打开

HPjmeter集成了以前的HPjtune功能，可以分析在HP机器上产生的垃圾回收日志文件