Broadview®
www.broadview.com.cn

深入理解 JVM & G1 GC

周明耀／著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

深入理解 JVM & G1 GC

周明耀 / 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

G1 GC 提出了不确定性 Region，每个空闲 Region 不是为某个固定年代准备的，它是灵活的，需求驱动的，所以 G1 GC 代表了先进性。

本书主要为学习 Java 语言的学生、初级程序员提供 GC 的使用参考建议及经验，着重介绍了 G1 GC。中国的软件开发行业已经有几十年了，从目前的行业发展来看，单纯的软件公司很难有发展，目前流行的云计算、物联网企业实际上是综合性 IT 技术的整合，这就需要有综合能力的程序员。本书作者力求做到知识的综合传播，而不是仅仅针对 Java 虚拟机和 GC 调优进行讲解，也力求每一章节都有实际的案例支撑。本书具体包括以下几方面：JVM 基础知识、GC 基础知识、G1 GC 的深入介绍、G1 GC 调优建议、JDK 自带工具使用介绍等。

通读本书后，读者可以深入了解 G1 GC 性能调优的许多主题及相关的综合性知识。读者也可以把本书作为参考，对于感兴趣的主题，直接跳到相应章节寻找答案。

图书在版编目（CIP）数据

深入理解 JVM ＆ G1 GC / 周明耀著. —北京：电子工业出版社，2017.6
ISBN 978-7-121-31468-1

Ⅰ. ①深… Ⅱ. ①周… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2017）第 097447 号

责任编辑：董　英
印　　刷：三河兴达印务有限公司
装　　订：三河兴达印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：15.5　　字数：337 千字
版　　次：2017 年 6 月第 1 版
印　　次：2017 年 6 月第 1 次印刷
印　　数：3000 册　　定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

序

这是我第一次为人写序，心中不免忐忑，引用余光中先生对于写序的感受：“我为人写序，于人为略而于文为详，用意也无非要就文本去探人本，亦即其艺术人格；自问与中国传统的序跋并不相悖，但手段毕竟不同了。”周明耀的书追求的是人本，我则尽力而为。

周明耀是我以前的同事，我们是认识十多年的朋友，同时我也是他婚礼的伴郎，见证了他的一步步成长。周明耀无论对工作、技术，或者社会生活，都有着自己独特的见解。他始终对技术充满了敬畏之心，学无止境，并付诸实践。因此，才有了这本书。

以我对 GC 的理解，想要学习 GC，首先需要理解为什么需要 GC。随着应用程序所应对的业务越来越庞大、复杂，用户越来越多，没有 GC 就不能保证应用程序的正常进行。而经常造成 STW 的 GC 又跟不上实际的需求，所以才会不断地尝试对 GC 进行优化。正如周明耀在文中描述的，HotSpot 有这么多的垃圾回收器（Serial GC、Parallel GC、Concurrent Mark Sweep GC），为什么还要发布 Garbage First（G1）GC？原因就是这个。

当今的商业模式，更多依赖市场的力量，不断淘汰旧的行业，把有限的资源让给那些竞争力更强、利润率更高的企业。类似地，硅谷也在不断淘汰过时的人员，从全世界吸收新鲜血液。经过半个多世纪的发展，在硅谷地区形成了只有卓越才能生存的文化理念。本着这样的理念，GC 承担了淘汰垃圾、保存优良资产的任务。正如周明耀所说，随着 G1 GC 的出现，GC 从传统的连续堆内存布局设计，逐渐走向不连续内存块，这是通过引入 Region 概念实现的，也就是说，由一堆不连续的 Region 组成了堆内存。其实也不能说是不连续的，只是它从传统的物理连续逐渐变为逻辑上的连续，这是通过 Region 的动态分配方式实现的，我们可以把一个 Region 分配给 Eden、Survivor、老年代、大对象区间、空闲区间中的任意一个，而不是固定它的作用，因为越是固定，越是呆板。

总的来说，本书对 Java GC 机制的分析深入浅出，是对大数据 Java 内存回收的优秀实践。读完茅塞顿开、受益匪浅。很多技术细节应用之后，对产品性能有明显提升。在此感谢周明耀的分享，希望他能够写出更多优秀的书籍。

华为南京研究所大数据产品部维护经理　吴骏

前　言

7 岁那年，当我合上《上下五千年》一套三册全书时，我对自己说，我想当个作家。这一晃 27 年了，等待了 27 年，我的第一本书《大话 Java 性能优化》在 2016 年 4 月正式面世，2016 年 8 月第二次印刷，感谢读者的厚爱。第一次印刷时出现一些错别字，请原谅编辑小姑娘，99 万字对她来说确实太多了，这是我的责任，未来一定尽全力避免。《深入理解 JVM & G1 GC》是我的第二本书，也即将面世。对于我的每一本书，我都怀着忐忑、惊喜的心情，就像第一次面对我的女儿“小顽子”，给她取这个小名，是希望她顽强到底，因为我相信，你若顽强到底，一切皆有可能。

我喜欢看书，每年购买的书接近 100 本，也喜欢技术积累。一直没有出书的想法，直到遇到了电子工业出版社的董老师，在深圳南湖的一席畅谈后，我决定做一位业余的技术作家，致力于中国软件开发行业的技术推广、普及、推动。

每年都要面试很多学生，我感觉中国的大学不太注重实际项目开发能力的培养，较为教条，这也是我的系统丛书首先从 Java 技术开始的原因，它更加接地气。本书主要为学习 Java 语言的学生、初级程序员提供 JVM 和 GC 的使用和优化建议及经验，力求做到知识的综合传播，而不是仅仅针对 Java 虚拟机调优进行讲解。本书具体包括以下几方面：JVM 基础知识、GC 基础知识、G1 GC 的深入介绍、G1 GC 调优建议、JDK 自带工具使用介绍等。

本书基于 JDK8，总的来说，没有一招鲜式的性能调优秘籍或包罗万象的性能百科，能让你摇身一变成为老练的 GC 性能调优专家。相当数量的 GC 性能问题还需要专门的知识技能才能解决。性能调优在很大程度上是一门艺术。解决的 GC 性能问题越多，技艺才会越精湛。我们不只要关心 GC 的持续演进，也要积极地去了解它的设计原理和设计目标。

最后，自我介绍一下，我叫周明耀，研究生学历，12 年工作经验，IBM 开发者论坛专家作

者。我是一名 IT 技术狂热爱好者，一名九三学社社员，一名顽强到底的工程师。我推崇技术创新、思维创新，对于新技术非常的热爱。

感谢我的家人，和谐的家庭帮助我完成了这本书。我的妻子，她美丽、细心、博学、偶尔不那么温柔，但是我很爱她。我的小顽子，她天生性格很像我，希望她能够踏踏实实做人，保持创新精神，平平安安、健健康康地生活下去。感谢我妻子的父母、我的父母，他们帮我照顾小孩，我才有时间编写此书。感谢浙江省特级教师、杭州高级化学老师郑克良老师，郑老师的一句永远不要放弃，推动着我多年的发展。感谢数学老师张老师在公开场合对我智商的褒奖，第一次收获这样的赞赏，对我这样内向的孩子是多么的重要，谢谢。

这本书献给我记忆中的爷爷奶奶、外公外婆，你们给我的都是最美的回忆。

我相信这本书不是终点，它是麦克叔叔此生一系列技术书籍的一员，咱们下一本书见。

目　录

1

第 1 章

JVM & GC 基础知识

单细胞生物在地球上经历了几十亿年的进化过程，又过了几十亿年，细胞与周围的细胞相互接触，从而形成了活的球形有机体，单细胞生物才进化为多细胞生物。在最初的时候，球形是多细胞生命能够生存的唯一形状，因为细胞之间只有相互接近才能互相协调功能。又过了十亿年，生命终于进化出了第一个神经元细胞，它是一种纤细的条状细胞，能够使两个细胞即使相隔一段距离仍能够通信。正是这项激活创新的诞生，各种各样的生命开始繁荣发展。有了神经元，生命不再局限于球状，细胞可以组成任何形状、大小和功能。蝴蝶、兰花和袋鼠，各种各样的生命形态都变成了可能。生命瞬间拓展出了成千上万种可能性，繁荣到令人惊叹，直到美丽的生命无处不在。程序设计的学习就好像探索生命起源一样，你不了解起源，就不可能找到正确的方法。[1]

JVM 是 Java 语言可以跨平台、保持高发展的根本，没有了 JVM，Java 语言将失去运行环

[1] 推荐带孩子去看《冰川时代》这本系列动画电影，它不只是动画片，也是对于大自然的思考，中国动画片倾向于培养人的三观，美国人的动画片更倾向于让人解释自己的行为，自己约束自己的行为。

境。针对 Java 程序的性能优化一定不可能避免针对 JVM 的调优，随着 JVM 的不断发展，我们的应对措施也在不断地跟随、变化，内存的使用逐渐变得越来越复杂。所有高级语言都需要垃圾回收机制的保护，所以 GC 就是这么重要。

本章主要介绍和解决以下问题，这些也是全书的基础。

- 为什么我们需要了解 JVM 和 GC，这是您阅读本书的依据。
- 了解 GC 的基础常用术语知识，作者和读者需要对术语定义进行统一。
- 了解 JVM 的基础知识，包括堆、栈、方法区等。
- 为深入了解 JVM 和 GC 做好知识储备。

1.1 引言

还记得机器猫吗？他和康夫有一张书桌，书桌的抽屉其实是一个时空穿梭通道，让我们操作机器猫[1]的时空机器，回到 1998 年。那年的 12 月 8 日，第二代 Java 平台的企业版 J2EE[2]正式对外发布。为了配合企业级应用落地，1999 年 4 月 27 日，Java 程序的舞台——Java HotSpot Virtual Machine（以下简称 HotSpot[3]）正式对外发布，并从这之后发布的 JDK1.3 版本开始，HotSpot 成为 Sun JDK 的默认虚拟机。

既然有了虚拟机，那一定需要收集垃圾的机制，这就是 Garbage Collection[4]，对应的产品我们称为 Garbage Collector，垃圾回收器。1999 年伴随 JDK1.3.1 一起来的是串行方式的 Serial GC[5]，它是第一款 GC，并且这只是起点。此后，JDK1.4 和 J2SE1.3 相继发布。2002 年 2 月 26 日，J2SE1.4 发布，Parallel GC[6]和 Concurrent Mark Sweep[7]（CMS）GC 跟随 JDK1.4.2 一起发布，并且 Parallel GC 在 JDK6 之后成为 HotSpot 默认 GC。

---

[1] 由日本漫画家藤本弘（笔名藤子・F・不二雄）和安孙子素雄（笔名藤子不二雄 A）共同创作的漫画作品。笔者最喜欢的动画片。

[2] 全称 Java 2 Platform Enterprise Edition，是一套全然不同于传统应用开发的技术架构，包含许多组件，主要可简化且规范应用系统的开发与部署，进而提高可移植性、安全与再用价值。

[3] 来源于无线热点。原指在公共场所提供无线局域网（Wi Fi）接入 Internet 服务的地点。

[4] 其实每种高级语言都有这个机制，不单单是针对 Java 语言的，推荐大家阅读 Richard Jones 等人编著的 *The Garbage Collection Handbook* 这本书，最好读英文原版的。

[5] 一种单线程的 GC。

[6] 与 Serial GC 一样都基于分代概念，差别是它采用了多线程方式加速运行垃圾回收。

[7] 是一款让应用程序可以和 GC 分享处理器资源的设计落地产品。

HotSpot 有这么多的垃圾回收器，那么如果有人问，Serial GC、Parallel GC、Concurrent Mark Sweep GC 这三个 GC 有什么不同呢？

- 如果你想要最小化地使用内存和并行开销，请选 Serial GC。
- 如果你想要最大化应用程序的吞吐量，请选 Parallel GC。
- 如果你想要最小化 GC 的中断或停顿时间，请选 CMS GC。

那么问题来了，既然我们已经有了上面三个强大的 GC，为什么还要发布 Garbage First（G1）GC 呢？我只能说：“人类的追求是无限的，就像女人永远追求更美丽。”

为什么名字叫作 Garbage First（G1）呢？故事背景稍微讲解一下，具体的内容请读者看后续章节。因为 G1 是一个并行回收器，它把堆内存分割为很多不相关的区间（Region），每个区间可以属于老年代或者年轻代，并且每个年龄代区间可以是物理上不连续的。老年代区间这个设计理念本身是为了服务于并行后台线程，这些线程的主要工作是寻找未被引用的对象[1]，而这样就会产生一种现象，即某些区间的垃圾（未被引用对象）多于其他的区间。我们后面会介绍，垃圾回收时都是需要停下应用程序的，不然就没有办法防止应用程序的干扰[2]，然后 G1 GC 可以集中精力在垃圾最多的区间上，并且只费一点点时间就可以清空这些区间里的垃圾，腾出完全空闲的区间。绕来绕去终于明白了，由于这种方式的侧重点在于处理垃圾最多的区间，所以我们给 G1 一个名字：垃圾优先（Garbage First）。

G1 内部主要有四个操作阶段，这四个阶段也是第 4 章和第 5 章的主要内容，即：

- 年轻代回收（A Young Collection）；
- 运行在后台的并行循环（A Background，Concurrent Cycle）；
- 混合回收（A Mixed Collection）；
- 全量回收（A Full GC）。

1.2　基本术语

结合我上一本书出版后的反馈信息以及个人阅读专业书籍的经验，由于计算机程序语言全部都是由国外创造、开发的，所以很多单词是经中国的技术专家由英文单词翻译的，这样可能的结果是专业术语按照自己的理解翻译，版本会较多，就如同各地方言一样。我觉得本书的专

[1] 在 GC 看来，未被引用对象是可以被回收的。

[2] 因为应用程序会不断地生产垃圾。警察办案时不是也要清场嘛，原因是一致的。

业性较强，很多英文单词容易出现歧义，所以我觉得有必要在这里统一一下对专业术语的中英文对照，以及对应的解释。我翻译和理解得不一定很到位，所以事先统一中英文对照还是很有必要的。

另外，需要提前说明的是，为了让每一个章节相对独立，有利于读者跳过一些章节，所以一些知识可能会重复介绍，这不是失误，是为了实现我对于我所有出版读物的规划，让技术以实践为目标，让使用更简单、更易落地。

1.2.1 Java 相关术语

1.2.1.1 Millisecond

GC 内部的动作（某一个过程）一般都是在毫秒级完成。毫秒（ms）是一种较为微小的时间单位，是一秒的千分之一。除了毫秒以外，还有皮秒、纳秒、微秒，计算公式分别如下：

- 皮秒，符号 ps（英语：picosecond，1 皮秒等于一万亿分之一秒（10^{-12} 秒）

1,000 皮秒 = 1 纳秒；1,000,000 皮秒 = 1 微秒；1,000,000,000 皮秒 = 1 毫秒；1,000,000,000,000 皮秒 = 1 秒。

- 纳秒，符号 ns（英语：nanosecond ）

1 纳秒等于十亿分之一秒（10^{-9} 秒）；1 纳秒 = 1000 皮秒；1,000 纳秒 = 1 微秒；1,000,000 纳秒 = 1 毫秒；1,000,000,000 纳秒 = 1 秒。

- 微秒，符号 μs（英语：microsecond ）

1 微秒等于一百万分之一秒（10^{-6} 秒）；0.000 001 微秒 = 1 皮秒；0.001 微秒 = 1 纳秒；1,000 微秒 = 1 毫秒；1,000,000 微秒 = 1 秒。

- 毫秒，符号 ms（英语：millisecond ）

1 毫秒等于一千分之一秒（10^{-3} 秒）；0.000 000 001 毫秒 = 1 皮秒；0.000 001 毫秒 = 1 纳秒；0.001 毫秒 = 1 微秒；1000 毫秒 = 1 秒。

1.2.1.2 Megabyte

GC 的区间划分基数一般采用兆级别，所以这里需要解释 MB。

MB（全称 MByte）：计算机中的一种储存单位，读作“兆”。数据单位 MB 与 Mb（注意 B 字母的大小写）常被误认为是一个意思，其实 MByte 含义是“兆字节”，Mbit 的含义是“兆

比特”。MByte 是指字节数量，Mbit 是指比特位数。MByte 中的“Byte”虽然与 Mbit 中的“bit”翻译一样，都是比特，也都是数据量度单位，但二者是完全不同的。Byte 是“字节数”，bit 是“位数”，在计算机中每八位为一字节，也就是 1Byte=8bit，是 1∶8 的对应关系。因此在书写单位时一定要注意 B 字母的大小写和含义。

1.2.1.3　Java 应用程序如何配置 JVM 参数

这个实践方式需要贯穿整本书，毕竟只有学会配置参数（JVM 选项），才能查看 GC 日志，最终才能主导讨论 GC 优化方法。

这里不是说如何配置正确的参数，仅仅只是演示一下如何在 Eclipse[1]和 Linux[2]这两个普遍运行 Java 程序的场景如何配置 JVM 参数（后面都统一称为选项）。

Eclipse 工具的使用方法如下：

选择 Eclipse → Run → Run Configurations → Arguments → VM arguments，在左边 Java Application 栏选中要设置的 Project 运行类，在 VM arguments 中填入 -Xms64m -Xmx256m。

这里-Xms 是设置内存初始化的大小（如上面的 64m），而-Xmx 是设置最大能够使用内存的大小（如上面的 256m，最好不要超过物理内存）。配置方式如图 1-1 所示，演示代码如代码清单 1-1 所示，运行输出请见代码清单 1-2 和代码清单 1-3。

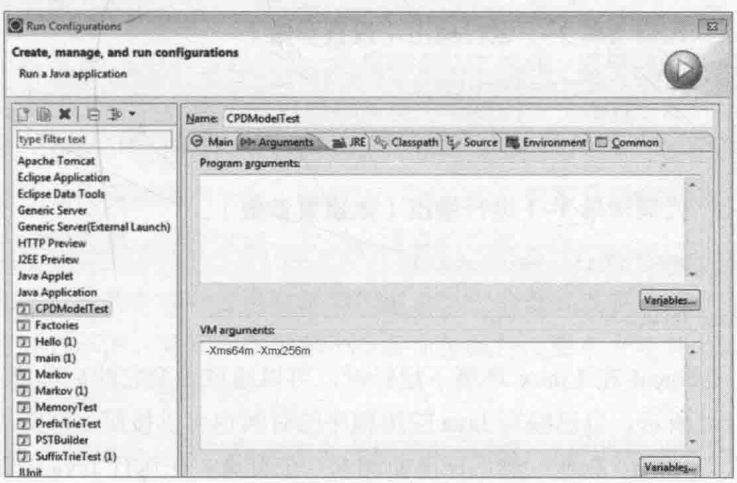

图 1-1　Eclipse 配置参数演示图

[1] 一种开源的 IDE 工具，原为 IBM 公司内部产品，下载地址 https://eclipse.org/downloads/。

[2] Linux 有一个蔚为壮观的生态系统，包括形形色色的贡献者、提供商和用户，这里泛指操作系统。

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　代码清单 1-1 运行输出（设置参数）

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

如果不配置选项，让 JVM 自己根据机器配置使用默认选项，输出如代码清单 1-3 所示。

代码清单 1-3　代码清单 1-1 运行输出（未设置参数）

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

下面介绍 Linux 场景。

举一个例子，Tomcat 在 Linux 环境下运行时，可以通过改变它的启动脚本来添加 JVM 参数，如代码清单 1-4 所示。自己编写 Java 应用程序的时候也可以按照这样的思路，编写一个脚本，可以是 Shell、Python、Perl[1]，然后在里面加入一个变量，在运行 Java 关键字的时候将这个关键字加为运行属性。

[1] Linux 的脚本三兄弟。