HZ BOOKS
华章IT

由资深Java工程师撰写，来自一线实践经验的结晶

详细解剖ZGC的运行原理以及调优方法，讲解细腻，图示丰富，可帮助Java工程师深入理解垃圾回收技术

Java
核心技术
系列

新一代垃圾回收器

ZGC设计与实现

ZGC Implementation and Performance Tuning

彭成寒 著

机械工业出版社
China Machine Press

Java 核心技术系列

新一代垃圾回收器
ZGC设计与实现

ZGC Implementation and Performance Tuning

彭成寒 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

新一代垃圾回收器 ZGC 设计与实现 / 彭成寒著 . —北京：机械工业出版社，2019.7
（Java 核心技术系列）

ISBN 978-7-111-63365-5

I. 新… II. 彭… III. 内存贮器 – 计算机算法 IV. TP333.1

中国版本图书馆 CIP 数据核字（2019）第 160526 号

新一代垃圾回收器 ZGC 设计与实现

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：赵亮宇 责任校对：李秋荣
印 刷：北京诚信伟业印刷有限公司 版 次：2019 年 9 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：13.25
书 号：ISBN 978-7-111-63365-5 定 价：89.00 元

客服电话：（010）88361066 88379833 68326294 投稿热线：（010）88379604
华章网站：www.hzbook.com 读者信箱：hzit@hzbook.com

PREFACE

前　言

JDK 11 于 2018 年 9 月 25 日正式发布，这个版本引入了许多新的特性，其中最为引人注目的就是实现了一款新的垃圾回收器 ZGC。Java 开发人员日常工作中最关注、接触最多的就是 JVM 中的垃圾回收器，所以该垃圾回收器一经发布，立即吸引了大量开发人员的目光。在 JDK 11 中，ZGC 被明确标记为实验性质（意味着还不成熟），这样一款尚不成熟的垃圾回收器为什么能合入 OpenJDK 的官方项目中？它对以前的垃圾回收器的改进体现在哪里？它的创新点是什么？它的不足有哪些？本书尝试从 ZGC 的代码出发，分析 ZGC 的设计和实现，希望能找到上述问题的答案。

ZGC 是一款开源的垃圾回收器，本书从原理和代码角度对 ZGC 进行剖析，与大家一起学习 ZGC，并希望通过本书的介绍让更多的人认识和使用它，也希望大家在学习的过程中都能理解、掌握、精通 ZGC，并能在社区中贡献自己的力量。

本书共分为 10 章：

- 第 1 章介绍 JVM 中实现的垃圾回收器，其中着重介绍了 G1，最后介绍了 ZGC 对 G1 的改进以及当下 ZGC 尚需完善之处。
- 第 2 章首先介绍内存地址多视图映射，然后介绍 ZGC 中的物理内存和虚拟内存，以及它们的管理，最后介绍 ZGC 如何分配对象。
- 第 3 章主要介绍 ZGC 中涉及的四大控制线程：ZDirector 负责垃圾回收的触发，ZDriver 负责垃圾回收的执行，ZStat 负责收集统计信息，VMThread 负责控制进行 STW 操作。
- 第 4 章介绍 ZGC 如何利用地址多视图映射设计并发算法进行并发标记、并发转移和并发重定位。
- 第 5 章介绍 ZGC 垃圾回收过程的 10 个步骤以及每一步所做的工作，同时给出了算法示例图演示整个垃圾回收的过程。
- 第 6 章分析一个完整的 ZGC 运行日志，并针对每一行日志进行解释，为读者了解

ZGC 的运行情况提供帮助。

- 第 7 章首先介绍 ZGC 中最常用的参数，包括 ZGC 新引入的参数、ZGC 重用的通用 GC 参数，然后介绍分别使用 G1 和 ZGC 作为垃圾回收器运行 Cassandra 和 YCSB，从停顿时间和吞吐率两个方面比较 ZGC 和 G1 的运行效果。
- 第 8 章主要介绍 ZGC 目前存在的不足以及未来的发展方向。
- 第 9 章介绍两种调试方法：根据源代码编译后使用 GDB 调试 JVM，着重介绍 ZGC 垃圾回收过程的调试；根据 HotSpot Debugger 工具对运行的 Java 程序进行分析。
- 第 10 章对 Shenandoah 进行简要介绍。Shenandoah 在 JDK 12 中作为实验项目加入 OpenJDK，它和 ZGC 的定位非常类似，但实现方法并不相同。该章简单地介绍 Shenandoah 和 G1、ZGC 之间的区别，Shenandoah 垃圾回收触发的策略以及 Shenandoah 实现的几种垃圾回收算法。

本书主要基于 JDK 11 源代码进行分析，所用的版本是 jdk11u1，读者可以自行到 OpenJDK 的官网下载，也可以从笔者在 GitHub（https://github.com/chenghanpeng/jdk11u）的备份中下载。

在本书中，为了能够让读者更加清晰、直观地了解一些基本概念，笔者设计了一些样例程序，这些样例代码可以从仓库（https://github.com/chenghanpeng/jdk11u/tree/master/example）中下载。另外，本书介绍了 ZGC 在 JDK 12 中的新功能——类回收，为了便于读者学习研究，笔者也维护了一份 JDK 12 的源代码（https://github.com/chenghanpeng/jdk12），供感兴趣的读者下载。

最后再强调一点，ZGC 处于持续迭代开发中，变化也会很快。为了能够深入探索 ZGC，希望读者在阅读本书时要始终抱着质疑的态度，不断地问自己：书中的介绍和解释是否正确？ZGC 的实现是否有改进的空间？如果有该如何改进？只有不断地提出问题、解决问题，才能深入理解和运用 ZGC。

由于笔者水平有限，时间仓促，书中难免存在错漏之处，恳请读者批评指正。你可以通过 https://github.com/chenghanpeng/jdk11u/issues 提交问题。期待能够得到读者朋友们的真情反馈，在技术道路上互勉共进。

在创作本书的过程中，得到了很多朋友以及同事的帮助和支持，在此表示衷心的感谢！

感谢策划编辑吴怡的支持和鼓励，她不仅给出了非常多的写作意见和建议，还不厌其烦地、认真地和笔者沟通，力争做到清晰、准确、无误地将内容呈现给读者。

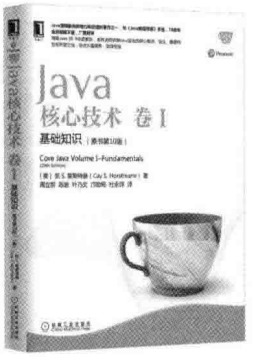
感谢我的家人，特别是我的儿子，能够体谅我牺牲了陪伴他的时间。有了他们的支持和帮助，我才有时间和精力去完成写作。

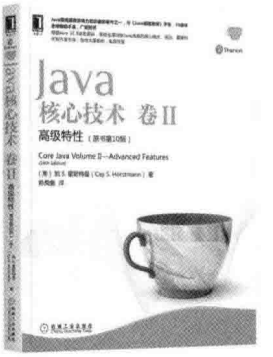
推荐阅读

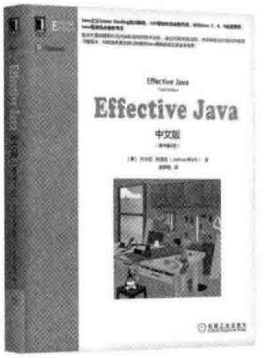

Java语言程序设计
与数据结构
（基础篇）


Java语言程序设计
与数据结构
（进阶篇）


Java程序设计概念
对象先行

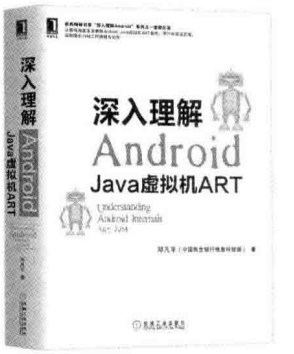

Java
核心技术 卷I
基础知识


Java
核心技术 卷II
高级特性

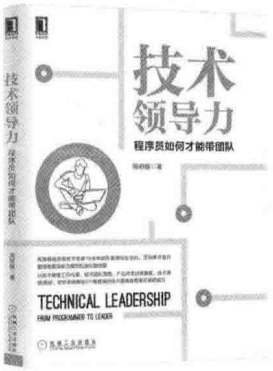

Effective Java
中文版

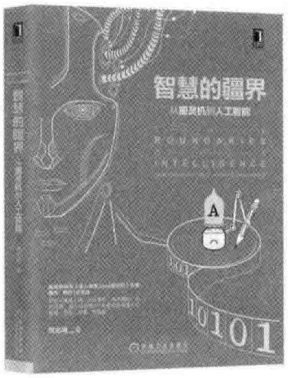

Java
并发编程的艺术

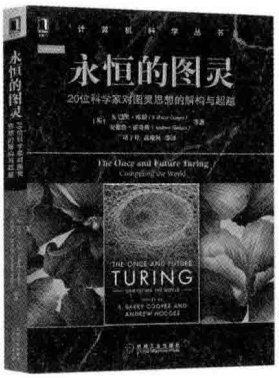

自己动手写
Java虚拟机


深入理解
Android
Java虚拟机ART

推荐阅读







技术领导力

作者是海康威视高级技术专家，海康威视是上市公司，市值曾超过4000亿，是AI和安防领域的龙头企业。

作者有超过10年的技术团队管理经验。

本书从技术管理工作内涵、技术团队管理、产品开发过程管理、技术调研/预研、软件系统架构5个维度阐述技术管理者需要具备的能力。

本书为程序员晋升为管理者提供了能力模型和进化路线图，同时为日常的管理工作提供了指导。

智慧的疆界

每一位程序员都应该了解人工智能，学习人工智能这本书是公认的首选。

这是一部对人工智能充满敬畏之心的匠心之作，《深入理解Java虚拟机》作者耗时一年完成，它将带你从奠基人物、历史事件、学术理论、研究成果、技术应用等5个维度全面读懂人工智能。

本书以时间为主线，用专业的知识、通俗的语言、巧妙的内容组织方式，详细讲解了人工智能这个学科的全貌、能解决什么问题、面临怎样的困难、尝试过哪些努力、取得过多少成绩、未来将向何方发展，尽可能消除人工智能的神秘感，把阳春白雪的人工智能从科学的殿堂推向公众面前。

永恒的图灵

图灵诞辰百年至今，伟大思想的光芒恒久闪耀。本书云集20位不同方向的顶尖科学家，共同探讨图灵计算思想的滥觞，特别是其对未来的重要影响。这些内容不仅涵盖我们熟知的计算机科学和人工智能领域，还涉及理论生物学等并非广为人知的图灵研究领域，最终形成各具学术锋芒的15章。如果你想追上甚至超越这位谜一般的天才，欢迎阅读本书，重温历史，开启未来。

CONTENTS

目　录

01 CHAPTER

第1章 垃圾回收器概述

Java是流行多年的编程语言，深受广大程序员的欢迎，其最主要的两个特点为：

- 跨平台："一次编译，到处运行"是最为贴切的总结。一次编译指的是Java源文件被编译器编译成字节码文件（通常以.class作为后缀，所以也称为Class文件）；到处运行指的是在安装了Java Virtual Machine（JVM）的不同平台上，Class文件都可以运行，由JVM负责解释或者编译执行字节码。
- 垃圾回收：程序员不用再像C/C++程序员一样关心内存的分配和释放，由垃圾回收器负责内存的管理，所以提高了程序开发的效率，减少了内存泄漏的概率。垃圾回收器由JVM的后台线程实现垃圾对象[⊖]的识别和回收。

自Java中引入垃圾回收器以来，垃圾回收器的发展从未停止过。Java中成熟的垃圾回收器有串行垃圾回收器、并行垃圾回收器、并发标记回收器（Concurrent Mark Sweep，CMS）、垃圾优先回收器（Garbage First，也称为G1）。在JDK 11中引入了一款新的垃圾回收器ZGC，在JDK 12中又引入了另一款新的垃圾回收器Shenandoah。虽然新的垃圾回收器不断地涌现，但是垃圾回收的基本算法变化并不大。简单来说，回收算法主要有复制、标记清除、标记压缩。JVM中不同的垃圾回收器都是基于这些基本算法实现的，不同的垃圾回收器区别在于：选择的算法不同，实现时后台线程采用的并行/并发方式不同。

本章介绍JVM中实现的垃圾回收器，并着重回顾了G1，最后介绍为什么需要ZGC，以及ZGC的创新点和不足之处。

⊖ 垃圾对象指的是应用程序中分配、使用后不再使用的对象。

1.1　垃圾回收算法

Garbage Collection（GC）垃圾回收（垃圾收集）指的是程序不关心对象在内存中的生存周期，创建后只需要使用，不用关心何时释放以及如何释放，由 JVM 自动管理内存、释放这些对象所占用的空间。垃圾回收的历史非常悠久，从 1960 年 Lisp 语言开始就支持垃圾回收。垃圾回收针对的是堆空间，目前垃圾回收算法主要有两类：

- 引用计数法：在堆内存中分配对象时，会为对象分配一段额外的空间，这个空间用于维护一个计数器，如果对象增加了一个新的引用，则将增加计数器的值；如果一个引用关系失效，则减少计数器的值。当一个对象的计数器的值变为 0，则说明该对象已经被废弃，处于不活跃状态，可以被回收。引用计数法需要解决循环依赖的问题，大家熟知的 Python 语言中的垃圾回收就使用了引用计数法。
- 可达性分析法（也称为根引用分析法），基本思路就是通过根集合（root set）作为起始点，从这些节点出发，根据引用关系开始搜索，所经过的路径称为引用链，当一个对象没有被任何引用链访问到时，则证明此对象是不活跃的，可以被回收。在 JVM 中常见的根（root）有线程栈帧（thread frame，用于跟踪线程中活跃对象）、符号表（symbol dictionary）、字符串表（string table）、对象监视器（object synchronizer）、元数据对象（universe）等，这些根共同构成了根集合。

这两种算法各有优缺点，具体可以参考其他文献。JVM 的垃圾回收采用了可达性分析法。垃圾回收算法也在不断地演化，按照不同的标准有不同的分类：

- 从垃圾回收算法实现主要分为复制（copy）、标记清除（mark-sweep）和标记压缩（mark-compact）。
- 从回收方式上可以分为串行回收、并行回收、并发回收。
- 从内存管理上可以分为代管理和非代管理。

关于垃圾回收的基本算法，本书不再介绍，具体可以参考其他书籍。

1.2　JVM 垃圾回收器

JVM 垃圾回收器基于分代管理和回收算法，结合回收的方式，实现了串行回收器、并行回收器、CMS、G1、ZGC 和 Shenandoah。这些垃圾回收器从程序执行方式的角度可以分为以下 3 类：

- 串行执行：应用程序和垃圾回收器交替执行，垃圾回收器执行的时候应用程序暂停执行。串行执行指的是垃圾回收器有且仅有一个后台线程执行垃圾对象的识别和回收。

- 并行执行：应用程序和垃圾回收器交替执行，垃圾回收器执行的时候应用程序暂停执行。并行执行指的是垃圾回收器有多个后台线程执行垃圾对象的识别和回收，多个线程并行执行。
- 并发执行：应用程序和垃圾回收器同时运行，除了在某些必要的情况下垃圾回收器需要暂停应用程序的执行，其余的时候在应用程序运行的同时，垃圾回收器的后台线程也运行，如标识垃圾对象并回收垃圾对象所占的空间。

Java 中的垃圾回收器对应的执行方式可以总结为表 1-1。

表 1-1　垃圾回收器对应的执行方式

执行方式	垃圾回收器
串行执行	串行垃圾回收器
并行执行	并行垃圾回收
并发执行	CMS、G1、ZGC、Shenandoah

下面我们看一下这些垃圾回收器的特点以及在执行垃圾回收时的活动图。

1.2.1　串行回收

使用单线程进行垃圾回收，在回收时应用程序（mutator）都需要执行暂停（Stop The World，STW）。新生代通常采用复制算法，老生代通常采用标记压缩算法。串行回收典型的执行过程如图 1-1 所示。

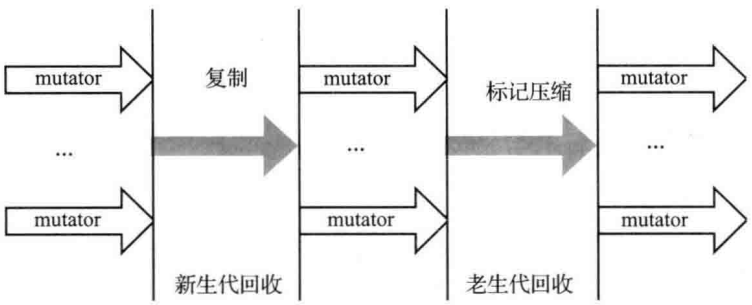


图 1-1　串行回收

注：本书的图例中没有 mutator 运行的区间都是指 STW。实际上串行回收中的老生代回收不仅仅回收老生代，还回收新生代。图中一个箭头表示一个线程，此图中执行垃圾回收过程只有一个箭头，表示只有一个后台线程执行回收任务。深色箭头表示的是垃圾回收工作线程，空心箭头表示应用程序线程。

1.2.2　并行回收

使用多线程进行垃圾回收，在回收时应用程序需要暂停，新生代通常采用复制算法，老生代通常采用标记压缩算法。并行回收的执行过程如图 1-2 所示。

在并发回收时，如果发现内存不足，需要对整个堆进行垃圾回收（也就是我们常说的 Full GC，也称为 FGC），在 Full GC 时需要 STW，并且是串行回收。

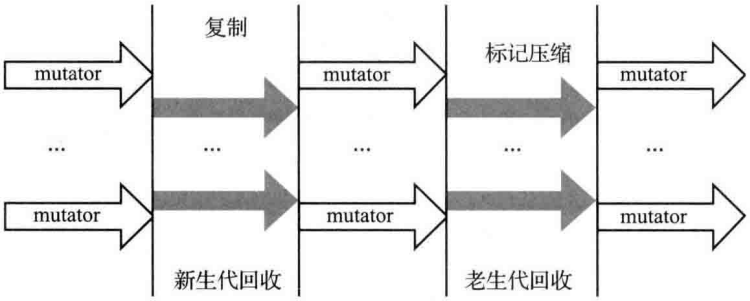


图 1-2　并行回收

1.2.3　CMS

整个回收期间划分成多个阶段：初始标记、并发标记、重新标记、并发清除等。在初始标记和重新标记阶段需要暂停应用程序线程，在并发标记和并发清除期间工作线程可以和应用程序并发运行。这个算法通常适用于老生代，新生代可以采用并行复制回收，也可以采用串行复制算法。CMS 垃圾回收的执行过程如图 1-3 所示。

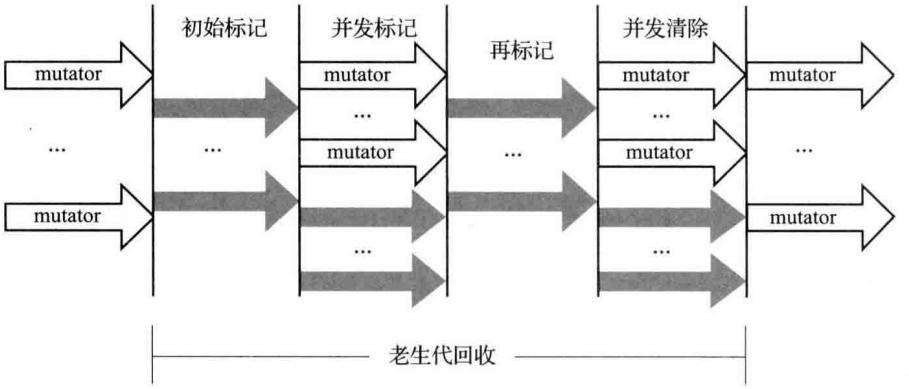


图 1-3　CMS 垃圾回收

同样，在老生代回收时，因为是并发执行，如果在分配内存时发现内存不足，则需要进行 FGC，也需要 STW 并对整个内存进行串行回收。

1.2.4　G1

从执行方式来看，垃圾回收器的发展经历了最初期的串行执行，到并行执行用于提高执行效率，再到目前主流的并发执行用于减少垃圾回收器停顿时间。第一款成熟的并发执行垃圾回收器是 CMS。CMS 是一款非常成功的垃圾回收器，是使用最多和最广的垃圾回收器，但是其复杂性（有上百个参数）给程序员的使用带来了不便，所以需要设计一款简单的垃圾回收器来替代 CMS，G1 应运而生。

G1 是从 JDK7 Update 4 及后续版本开始正式提供的。G1 致力于在多 CPU 和大内存服务器上对垃圾回收提供软实时目标（soft real-time goal）和高吞吐量（high throughput）。目前 G1 已经相当成熟，从众多的测评结果上看，也达到了 G1 最初的设计目标。从 JDK 9 开始 G1 作为默认的垃圾回收器，目前已经有不少公司开始在生产环境中逐步使用 G1。

G1 垃圾回收器的设计和前面提到的 3 种回收器都不同，在并行、串行以及 CMS 中针对堆空间的管理方式都是连续的，如图 1-4 所示。

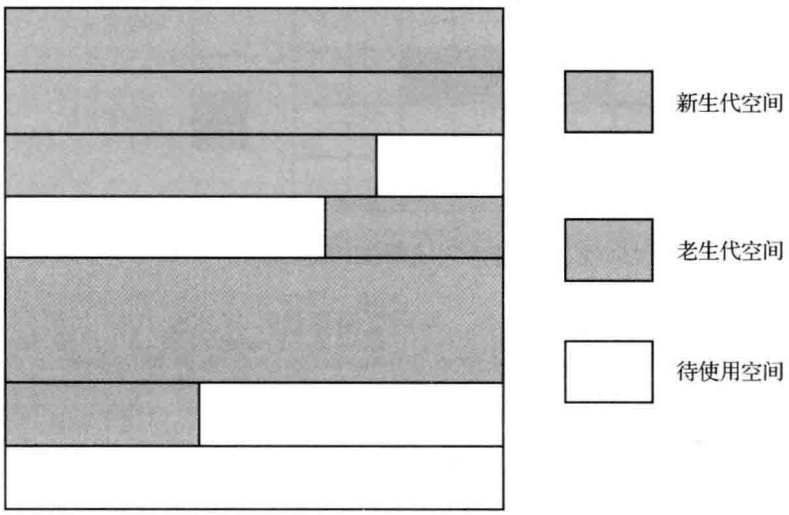


图 1-4　连续空间管理

连续的内存将导致垃圾回收时收集时间过长，停顿时间不可控。所以 G1 将堆拆成一系列的分区（heap region），这样在一个时间段内，大部分垃圾回收操作就只是针对一部分分区执行，而不是整个堆或整个（老年）代，从而满足在指定的停顿时间内完成垃圾回收的动作。G1 内存分区如图 1-5 所示。

在 G1 里，新生代就是一系列的内存分区，这意味着不用再要求新生代是一个连续的内存块。类似地，老生代也是由一系列的分区组成。在 JVM 运行时，从内存管理角度不需要预先设置分区是老生代分区还是新生代分区，而是在内存分配时决定：当新生代需

要空间时，则分区被加入新生代中；当老生代需要内存空间时，则分区被加入老生代中。事实上，G1 通常的运行状态是：映射 G1 分区的虚拟内存随着时间的推移在不同的代之间切换。例如，一个 G1 分区最初被指定为新生代，经过一次新生代的回收之后，整个新生代分区都被划入待使用的分区中，那它既可以作为新生代分区使用，也可以作为老生代分区使用。很可能在完成一个新生代回收之后，一个新生代的分区在未来的某个时刻被用于老生代分区。同样，在一个老生代分区完成回收之后，它就成为待使用分区，在未来某个时候作为一个新生代分区使用。

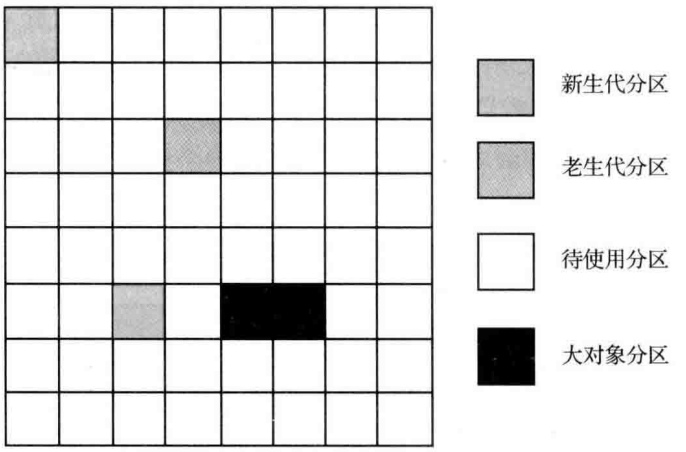


图 1-5　分区空间管理[⊖]

G1 新生代的回收方式是并行回收，采用复制算法。与其他 JVM 垃圾回收器一样，一旦发生一次新生代回收，整个新生代都会被回收。这就是我们常说的新生代回收（Young GC，YGC）。但是 G1 和其他垃圾回收器的不同之处在于：① G1 会根据预测时间动态地改变新生代的大小[⊜]；② G1 老生代的垃圾回收方式与其他 JVM 垃圾回收器对老生代处理有着极大的不同。G1 老生代的回收不会为了释放老生代的空间而对整个老生代进行回收。相反，在任意时刻只有一部分老生代分区会被回收，并且这部分老生代分区将在下一次增量回收时与所有的新生代分区一起被回收，这就是我们所说的混合回收（Mixed GC）。在选择老生代分区时，优先考虑垃圾多的分区。

老生代分区的选择涉及 G1 的并发标记算法，这个过程称为“并发标记阶段”。并发

⊖ 图 1-5 中，大对象分区也是老生代分区的一种，当对象比较大（超过一定大小）时被分配到大对象分区，一个大对象分区可以由多个分区组成。

⊜ 注意其他垃圾回收新生代的大小也可以动态地变化，但这个变化主要是根据内存的使用情况进行的。G1 中则是以预测时间为导向，根据内存的使用情况，调整新生代分区的数目。

标记是指并发标记线程和应用程序线程同时运行，它有 4 个典型的子阶段：初始标记子阶段、并发标记子阶段、再标记子阶段和清理子阶段。

1. 初始标记子阶段

负责标记所有从根集合直接可达的对象，根集合是对象图的起点，初始标记需要将应用程序线程暂停，也就是需要一个 STW 的时间段。在混合回收中的初始标记子阶段和新生代的初始标记几乎一样。实际上混合回收的初始标记子阶段是借用了新生代回收的结果，即新生代垃圾回收后的新生代 Survivor 分区作为根，所以混合回收一定发生在新生代回收之后，且不需要再进行一次初始标记。这就是所谓的“借道”。

2. 并发标记子阶段

当 YGC 执行结束之后，如果发现满足并发标记的条件，并发线程就开始进行并发标记。根据新生代的 Survivor 分区开始并发标记。并发标记的时机是在 YGC 后，只有内存消耗达到一定的阈值后才会触发。在 G1 中，这个阈值通过参数 InitiatingHeapOccupancyPercent 控制（默认值是 45，表示的是当已经分配的内存加上本次将分配的内存超过内存总容量的 45% 时就可以开始并发标记）。多个并发标记线程启动，每个线程每次只扫描一个分区，从而标记出存活对象。在标记的时候还会计算存活对象的数量，同时会计算存活对象所占用的内存大小，并计入分区空间。

并发标记子阶段会对所有分区的对象进行标记。这个阶段并不需要 STW，故标记线程和应用程序线程并发运行。使用 Snapshot-At-The-Beginning(SATB) 算法进行并发标记。

3. 再标记子阶段

再标记是最后一个标记阶段。在该阶段中，G1 需要一个 STW 的时间段，找出所有未被访问的存活对象，同时完成存活内存数据计算。引入该阶段是为了能够达到结束标记的目标。要结束标记过程，需要满足 3 个条件：

- 从根（survivor）出发并发标记子阶段已经标记出所有的存活对象。
- 标记栈是空的。
- 所有的引用变更对象都被处理了。这里的引用变更对象包括新增空间分配的对象和引用变更对象，新增空间所有对象被认为都是活跃的（即使对象已经“死亡”也没有关系，在这种情况下只是增加了一些浮动垃圾），引用变更处理的对象通过一个队列记录，在该子阶段会处理这个队列中所有的对象。

前两个条件是很容易满足的，但是满足最后一个条件是很困难的。如果不引入一个 STW 的再标记过程，那么应用会不断地更新引用，也就是说，会不断地产生新的引用变更，因而永远无法达成完成标记的条件。

这个子阶段是并行执行的。

4. 清理子阶段

再标记子阶段之后是清理子阶段，该子阶段也需要一个 STW 的时间段。清理子阶段主要执行以下操作：

- 统计存活对象，统计的结果将会用来排序分区，以用于下一次的垃圾回收时分区的选择。
- 交换标记位图，为下次并发标记做准备。
- 把空闲分区放到空闲分区列表中。这里的空闲分区指的是全都是垃圾对象的分区，如果分区中还有活跃对象，则不会释放，真正释放的动作发生在混合回收中。

该阶段比较容易引起误解的地方在于，清理子阶段并不会清理垃圾对象，也不会执行存活对象的复制。也就是说，在极端情况下，该阶段结束之后，空闲分区列表将毫无变化，JVM 的内存使用情况也毫无变化。

该子阶段也是并行执行的。

并发标记阶段完成之后，在下一次进行垃圾回收的时候就会回收垃圾比较多的老生代分区。这时进行的垃圾回收称为混合回收，混合回收和 YGC 最大的区别就是混合回收不仅仅回收所有的新生代分区，也回收部分垃圾多的老生代分区，所以 JVM 在实现混合回收时重用了 YGC 所有的代码，两者的不同之处就在于是否回收老生代分区。整个 G1 垃圾回收的过程如图 1-6 所示。

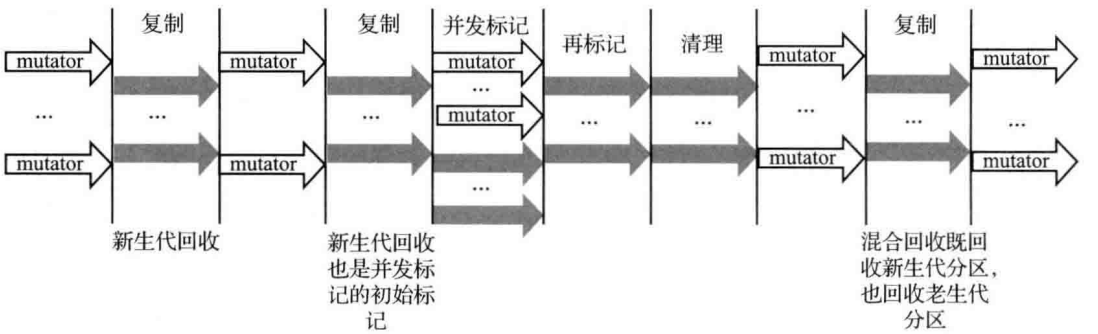


图 1-6　G1 垃圾回收


注意　在图 1-6 所示并发标记阶段中还可以发生 YGC（可以是一次 YGC，也可以是多次 YGC），但为了简化并未体现；另外，在图中混合回收也可能发生多次，因为 G1 对停顿时间是有要求的，G1 会根据预测的停顿时间决定一次回收老生代分区的数目，所以可能需要多次混合回收，才能完成并发标记阶段识别的垃圾比较多的老生代分区的回收。