Виды сборщиков мусора в Java

JSP & Servlets
18 уровень, 5 лекция

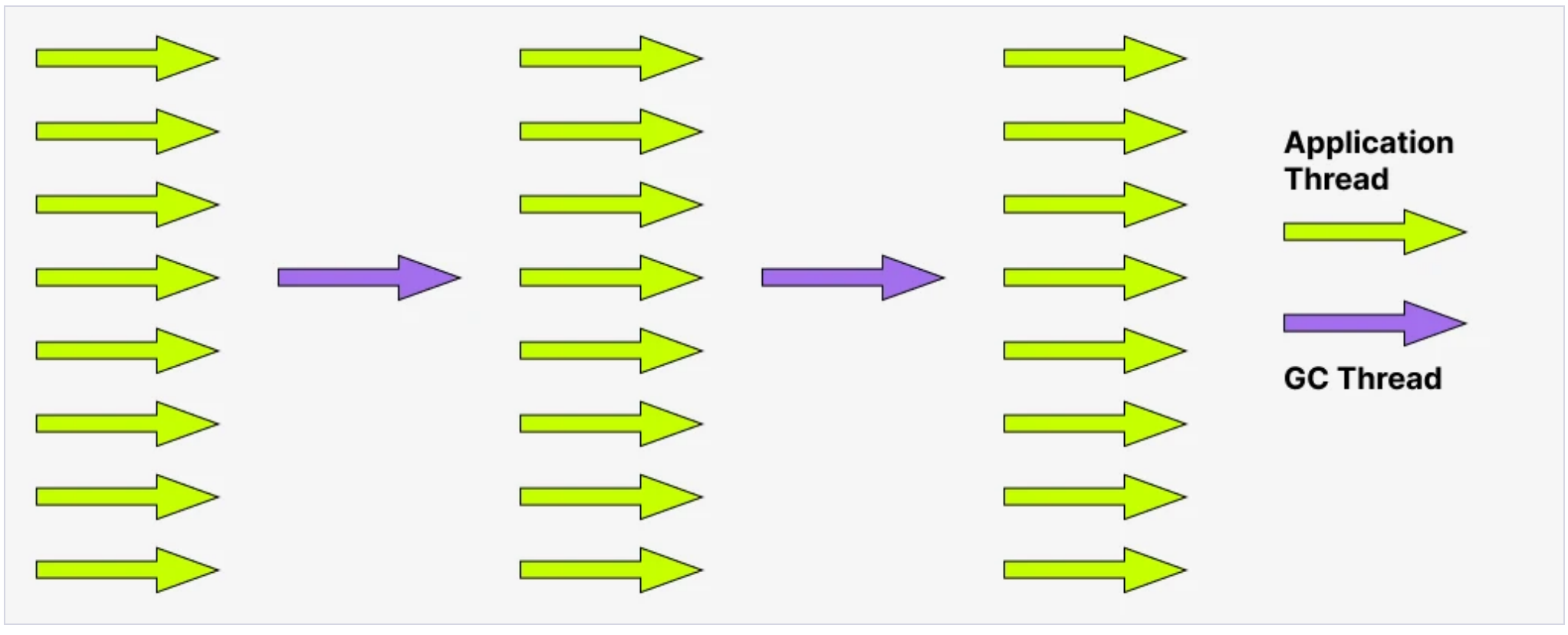
ОТКРЫТА

Serial GC

Сборка мусора повышает эффективность работы с памятью в Java, так как объекты, не имеющие ссылок удаляются из кучи и освобождается место для новосозданных объектов.

У Java виртуальной машины есть восемь типов сборщиков мусора. Рассмотрим каждый из них в деталях.

Серийный GC — это самая простая реализация GC. Она предназначена для небольших приложений, работающих в однопоточных средах. Все события сборки мусора выполняются последовательно в одном потоке. Уплотнение выполняется после каждой сборки мусора.



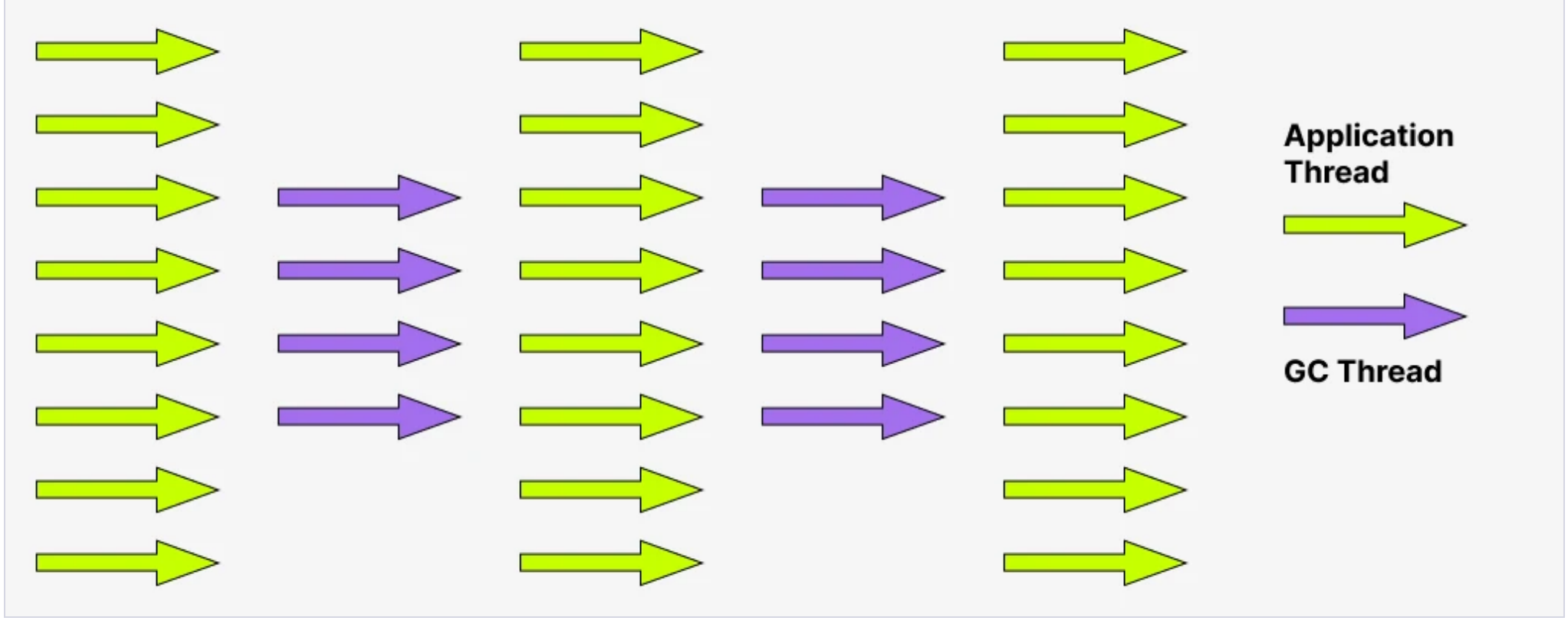
Запуск сборщика приводит к событию “остановки мира”, когда все приложение приостанавливает работу. Поскольку на время сборки мусора все приложение замораживается, не следует прибегать к такому в реальной жизни, если требуется, чтобы задержки были минимальными.

Аргумент JVM для использования последовательного сборщика мусора **-XX:+UseSerialGC**.

Parallel GC

Параллельный сборщик мусора предназначен для приложений со средними и большими наборами данных, которые выполняются на многопоточном или многопроцессорном оборудовании. Это реализация GC по умолчанию и она также известна как сборщик пропускной способности.

Несколько потоков предназначаются для малой сборки мусора в молодом поколении. Единственный поток занят основной сборкой мусора в старшем поколении.



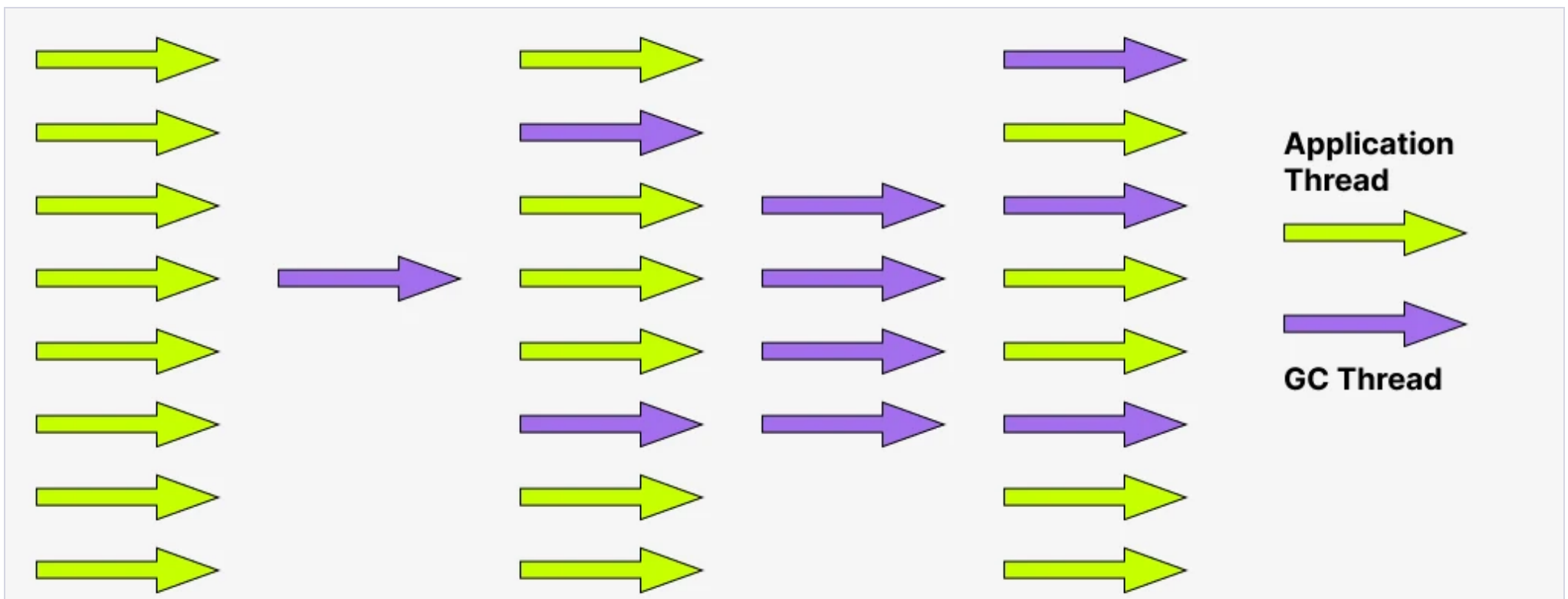
Запуск параллельного GC также вызывает “остановку мира” и приложение зависает. Такое поведение больше подходит для многопоточной среды, когда требуется завершить много задач и допустимы длительные паузы, например при выполнении пакетного задания.

Аргумент JVM для использования параллельного сборщика мусора: **-XX:+UseParallelGC**.

CMS GC

Также известен нам как **параллельный сборщик низких пауз**.

Тут для малой сборки мусора задействуются несколько потоков, происходит это через такой же алгоритм, как и в параллельном сборщике. Основная сборка мусора многопоточна, как и в старом параллельном GC, но CMS работает одновременно с процессами приложений, чтобы свести к минимуму события “остановки мира”.



Из-за этого сборщик CMS потребляет больше ресурсов процессора, чем другие сборщики. Если у вас есть возможность выделить больше ЦП для повышения производительности, то CMS предпочтительнее, чем простой параллельный сборщик. В CMS GC не выполняется уплотнение.

Аргумент JVM для использования параллельного сборщика мусора с разверткой меток: **-XX:+UseConcMarkSweepGC**.

G1 (Garbage first) GC

G1GC был задуман как замена CMS и разрабатывался для многопоточных приложений, которые характеризуются крупным размером кучи (более 4 ГБ). Он параллелен и конкурентен, как CMS, но “под капотом” работает совершенно иначе, чем старые сборщики мусора.

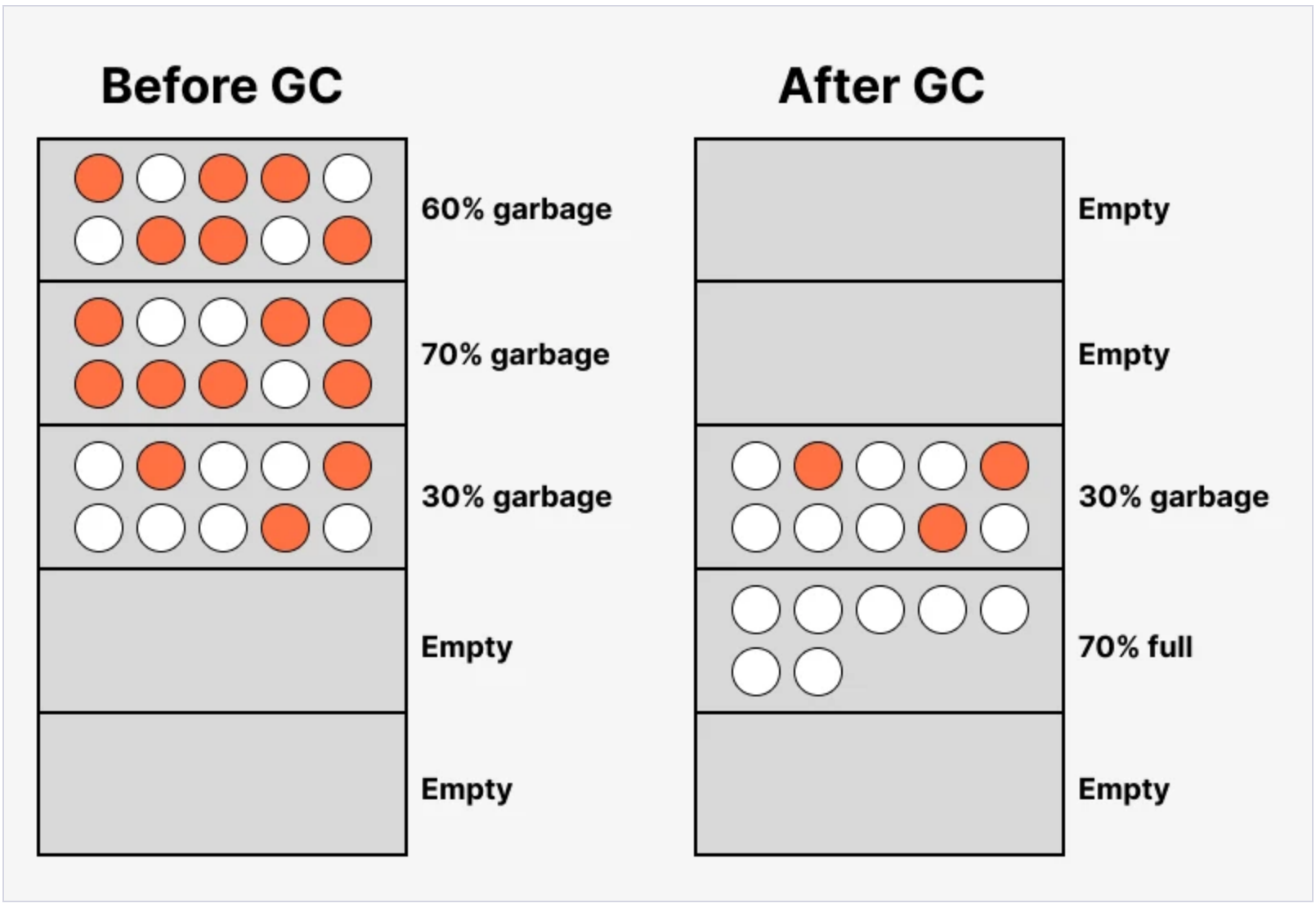
Хотя G1 также действует по принципу поколений, в нем нет отдельных пространств для молодого и старшего поколений. Вместо этого каждое поколение представляет собой набор областей, что позволяет гибко изменять размер молодого

поколения.

G1 разбивает кучу на набор областей одинакового размера (в зависимости от размера кучи) и сканирует их в несколько потоков. Область во время выполнения программы может неоднократно становиться как старой, так и молодой.

После завершения этапа разметки G1 знает, в каких областях содержится больше всего мусора. Если пользователь заинтересован в минимизации пауз, G1 может выбрать только несколько областей. Если время паузы неважно для пользователя или предел этого времени установлен высокий, G1 пройдет по большему числу областей.

Поскольку G1 GC идентифицирует регионы с наибольшим количеством мусора и сначала выполняет сбор мусора в них, он и называется: “Мусор — первым”.



Помимо областей Эдема, Выживших и Старой памяти, в G1GC присутствуют еще два типа.

- **Humongous** (Огромная) — для объектов большого размера (более 50% размера кучи).
- **Available** (Доступная) — неиспользуемое или не выделенное пространство.

Аргумент JVM для использования сборщика мусора G1: **-XX:+UseG1GC**.

Shenandoah (Шандара)

Shenandoah — новый GC, выпущенный как часть JDK 12. Ключевое преимущество Shenandoah перед G1 состоит в том, что большая часть цикла сборки мусора выполняется одновременно с потоками приложений. G1 может эвакуировать области кучи только тогда, когда приложение приостановлено, а Shenandoah перемещает объекты одновременно с приложением.

Shenandoah может компактировать живые объекты, очищать мусор и освобождать оперативную память почти сразу после обнаружения свободной памяти. Поскольку все это происходит одновременно, без приостановки работы приложения, то Shenandoah более интенсивно нагружает процессор.

Аргумент JVM для сборщика мусора Шенандоа: **-XX:+UnlockExperimentalVMOptions -XX:+UseShenandoahGC**.

ZGC

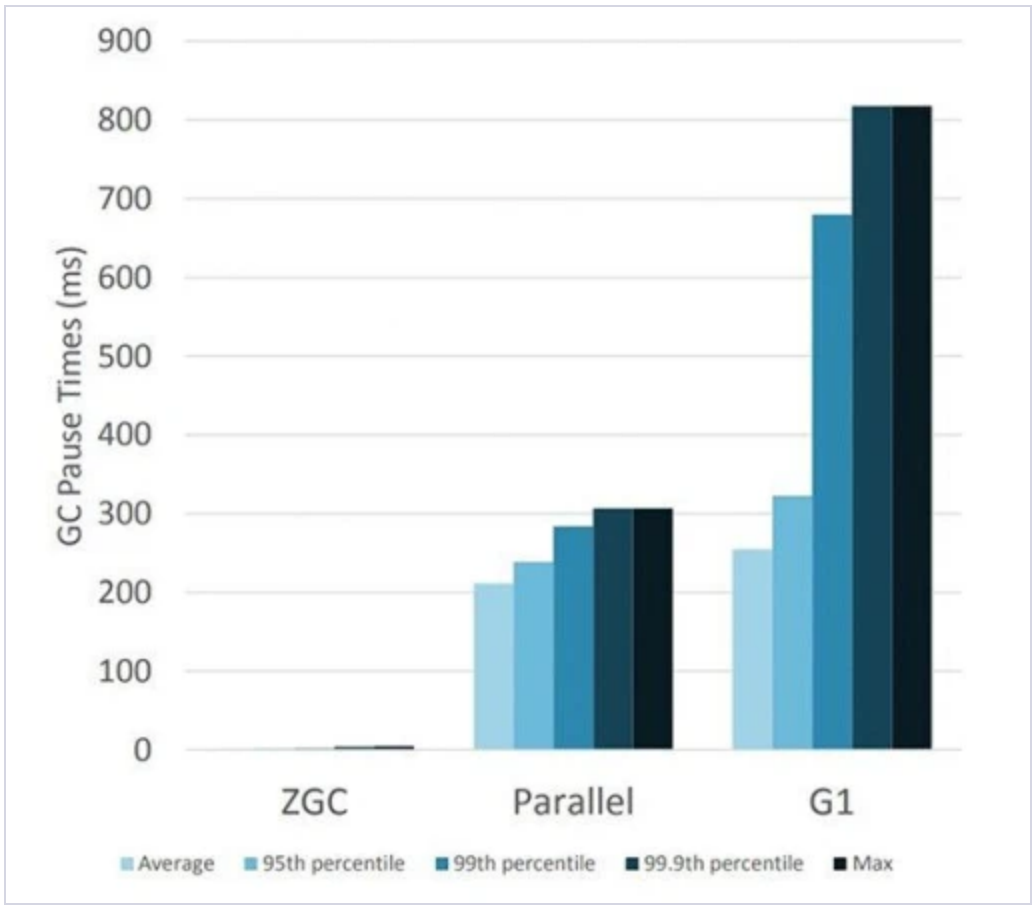
ZGC — еще один GC, выпущенный как часть JDK 11 и улучшенный в JDK 12.

Он предназначен для приложений, которые требуют быстродействия и низкой задержки (паузы в менее чем 10 мс) и/или задействуют очень большую кучу (несколько терабайт).

Основные цели ZGC— низкая задержка, масштабируемость и простота в применении. Для этого он позволяет Java-приложению продолжать работу, несмотря на то, что выполняются операции по сбору мусора. По умолчанию ZGC

освобождает неиспользуемую память и возвращает ее в операционную систему.

Таким образом, ZGC привносит значительное улучшение по сравнению с другими традиционными GC, обеспечивая чрезвычайно низкое время паузы (обычно в пределах 2 мс).



Аргумент JVM для использования сборщика мусора ZGC: **-XX:+UnlockExperimentalVMOptions -XX:+UseZGC**.

‹ Предыдущая лекция

Следующая лекция ›

+12

Комментарии (1) популярные новые старые

JavaCoder

Введите текст комментария

kv0ut Уровень 51 7 января, 23:01

"У Java виртуальной машины есть ВОСЕМЬ типов сборщиков мусора. Рассмотрим каждый из них в деталях."
И далее описывается ШЕСТЬ типов сборщиков

Ответить +2

RUSH

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.