sunshine4545

41 уровень

Минск

30.04.2020 3935 5

# Учимся гуглить | 4 уровень | 11 лекция




## 1. Как работает сборщик мусора в Java

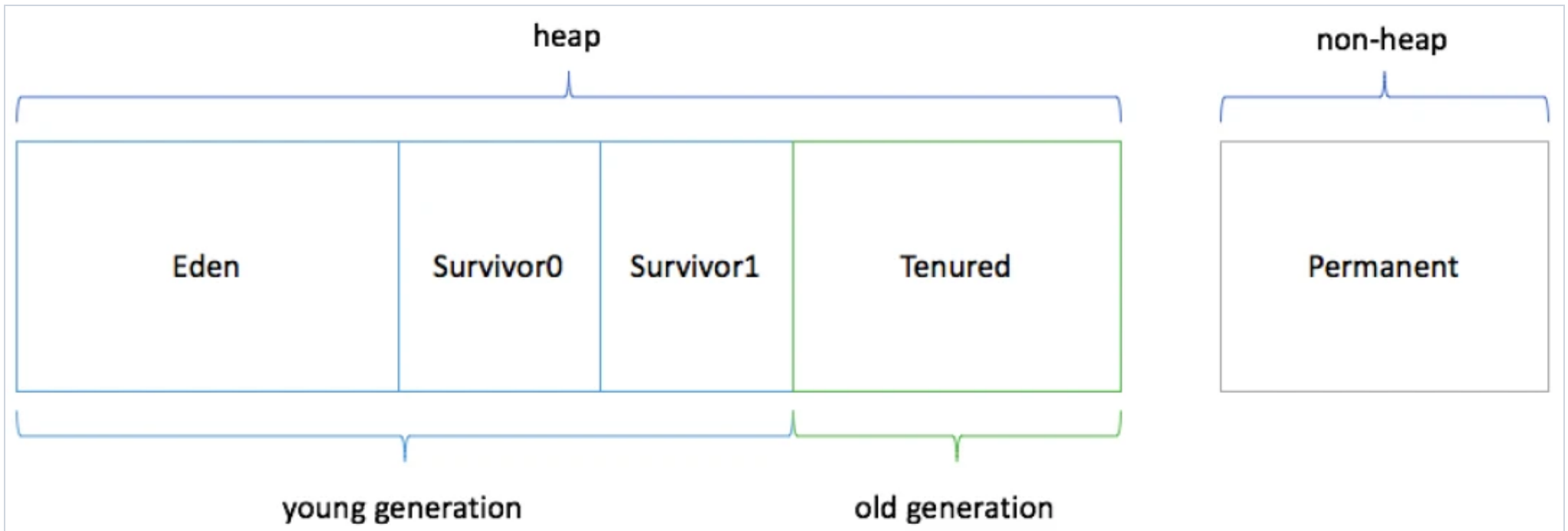
Сборщик мусора — это низкоприоритетный процесс, который запускается периодически и освобождает память, использованную объектами, которые больше не нужны. Сборщик мусора работает в фоновом режиме, параллельно с программой, в отдельном потоке. Основа для сборки мусора не подсчет ссылок, а разделение объектов на два вида — достижимые и недостижимые. Объект считается достижимым (живым), если на него ссылается другой достижимый (живой) объект. Достижимость считается от нитей. Работающие нити всегда считаются достижимыми (живыми), даже если на них никто не ссылается.

Все объекты в Java хранятся в специальной области памяти, которая называется куча (heap). Все объекты в программах можно разделить на два типа — условно говоря, простые объекты и “долгожители”.  “Долгожителями” считаются объекты, пережившие много сборок мусора. Чаще всего они будут существовать до конца работы программы. В итоге общая куча, где хранятся все созданные объекты, была разделена на несколько частей.

Первая часть имеет красивое название — Eden (библ. “Райский сад”), сюда объекты попадают после их создания. Именно в этой части выделяется память для новых объектов, когда мы пишем new. Объектов может создаться много, и когда в этой области заканчивается место, начинается первая, “быстрая” сборка мусора. Надо сказать, что сборщик мусора очень умен и выбирает алгоритм работы в зависимости от того, чего в куче больше — мусора или рабочих объектов. Если почти все объекты являются мусором, сборщик помечает “живые” объекты и переносит их в другую область памяти, после чего текущая

Область памяти, куда переносятся все объекты, пережившие хотя бы одну сборку мусора, называется Survival Space (“место для выживших”). Survival Space в свою очередь делится на поколения. Каждый объект относится к своему поколению в зависимости от того, сколько сборок мусора он пережил. Если одну — он относится к “Поколению 1”, если 5 — к “Поколению 5”. Вместе Eden и Survival Space образуют область под названием Young Generation (“молодое поколение”).

Помимо Young Generation в куче существует и другая область памяти — Old Generation (“старое поколение”). Сюда как раз попадают те самые объекты-долгожители, которые пережили много сборок мусора. Их выгоднее хранить отдельно от всех остальных. И только когда область Old Generation заполнена, т.е. даже объектов-долгожителей в программе так много, что памяти не хватает, производится полная уборка мусора. Она обрабатывает не одну область памяти, а вообще все созданные Java-машиной объекты. Естественно, она занимает куда больше времени и ресурсов. Именно поэтому объекты-долгожители было решено хранить отдельно. Когда место заканчивается в других областях, проводится так называемая “быстрая сборка мусора”. Она охватывает только одну область, и за счет этого является более экономичной и быстрой. В конце, когда забита уже даже область для долгожителей, в бой вступает полная уборка. Таким образом, самый “тяжеловесный” инструмент используется сборщиком только тогда, когда без этого уже не обойтись.




## 2. Какие бывают виды сборщиков мусора

Java has seven types of garbage collectors:

1. Serial Garbage Collector
2. Parallel Garbage Collector
3. CMS Garbage Collector
4. G1 Garbage Collector
5. Epsilon Garbage Collector
6. Z garbage collector
7. Shenandoah Garbage Collector

## 3. Что такое «поколения» объектов

Все объекты в Survival Space делятся на поколения. Каждый объект относится к своему поколению в зависимости от того, сколько сборок мусора он пережил. Если одну — он относится к “Поколению 1”, если 5 — к “Поколению 5”.

## 4. Для чего используется SoftReference

Объект, на который ссылаются только мягкие ссылки, может быть удален сборщиком мусора, если программе не хватает памяти. Если программе вдруг не хватает памяти, прежде чем выкинуть OutOfMemoryException, сборщик мусора удалит все объекты, на которые ссылаются мягкие ссылки и попробует выделить программе память еще раз. Объект, который удерживает от смерти только SoftReference может пережить сколько угодно сборок мусора и скорее всего, будет уничтожен при нехватке программе памяти.

## 5. Пример использования SoftReference

этого SoftReference. Если объекты, удерживаемые от смерти мягкими ссылками, будет занимать большую часть памяти, то сборщик мусора просто их поудаляет и все.

## 6. Пример использования WeakReference

Если на объект остались только слабые ссылки, то этот объект является живым, но он будет уничтожен при ближайшей сборке мусора. Объект, который удерживает от смерти только WeakReference не переживает ближайшей сборки мусора. Но пока она не произошла, его можно получить, вызвав метод `get()` у WeakReference и вызвать его методы или сделать что-нибудь еще. Пример использования WeakReference – это WeakHashMap.

## 7. Зачем нужен WeakHashMap

WeakHashMap – это HashMap, у которого ключи – это слабые ссылки – WeakReference. Ты хранишь в WeakHashMap пары объектов – ключ и значение. Но WeakHashMap ссылается на ключи не прямо, а через WeakReference. Поэтому, когда объекты, используемые в качестве ключей, станут слабодостижимыми, они уничтожатся при ближайшей сборке мусора. А значит, из WeakHashMap автоматически удалятся и их значения.
В WeakHashMap очень удобно хранить дополнительную информацию к каким-то объектам.

Во-первых, ее очень легко получить, если использовать сам объект в качестве ключа.

Во-вторых, если объект будет уничтожен, из HashMap исчезнет и он, и все привязанные к нему данные.

Например в программе есть нить, которая отслеживает работу некоторых объектов-заданий и пишет информацию о них в лог. Тогда эта нить может хранить отслеживаемые объекты в таком WeakHashMap. Как только объекты станут не нужны, сборщик мусора удалит их, автоматически удалятся и ссылки на них из WeakHashMap.

## 8. Что такое логгер

A Logger object is used to log messages for a specific system or application component.

Лог – это список произошедших событий. Чаще всего в лог пишется информация о параметрах метода, с которыми он был вызван, все перехваченные ошибки, и еще много промежуточной информации.

Весь процесс логирования состоит из трех частей.

- Первая часть – это сбор информации.
- Вторая часть – это фильтрование собранной информации.
- Третья часть – это запись отобранной информации.

## 9. Как настроить логгер

1. В pom.xml добавить зависимость.
2. Добавить в resources файл log4j.properties.
   Обычно настройки логгера log4j задаются в файле log4j.properties. В этом файле можно задать несколько appender'ов – объектов, в которые будут писаться данные.
3. Добавить в класс с бизнес-логикой `private static final Logger log = Logger.getLogger(xxx.class);`.



Использовалось:

- суперстатья
- статья

+82

## Комментарии (5)

популярные новые старые

**JavaCoder**

Введите текст комментария

**SSV** Уровень 27 — 5 апреля, 13:21

Отлично. Забрал в закладки.

Ответить 0

**Almielka** Уровень 37 — 13 апреля 2021, 15:52

В п.9 создаем объект с помощью LoggerFactory и его статического метода getLogger():

```
1  private static final Logger logger = LoggerFactory.getLogger(ClassName.class)
```

Ответить +1

**barracuda** Уровень 41, Санкт-Петербург, Россия EXPERT — 21 февраля 2021, 23:57

Да, все компактно собрал , и все по делу.

Ответить 0

**Татьяна** Уровень 41, Днепр, Украина — 9 декабря 2020, 22:22

Круто, большое спасибо!!!!

Ответить 0

**Alexey Prilessky** Уровень 40, Минск — 29 сентября 2020, 20:52

Ответить +1

JavaRush

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.

**ПОДПИСЫВАЙТЕСЬ**

**ЯЗЫК ИНТЕРФЕЙСА**

Русский