Столыпинский вестник

Научная статья
Original article
УДК 004.258

ПРЕИМУЩЕСТВА АВТОМАТИЗИРОВАННОГО УПРАВЛЕНИЯ ПАМЯТЬЮ НА ПРИМЕРЕ JAVA HOTSPOT

ADVANTAGES OF AUTOMATED MEMORY MANAGEMENT ON THE EXAMPLE OF JAVA HOTSPOT

Аникин Данила Алексеевич, Студент бакалавриата 3 курс,МИРЭА-Российский технологический университет (РТУ МИРЭА),119454, Россия, г. Москва, проспект Вернадского, 78,Институт информационных технологий, Россия, г. Москва

Anikin Danila Alekseevich, Bachelor's student 3rd year, MIREA-Russian Technological University (RCTU MIREA), 78 Vernadsky Avenue, Moscow, 119454, Russia, Institute of Information Technologies, Russia, Moscow

Аннотация: Настоящая статья посвящена обзору преимуществ и недостатков, которые присущи механизмам автоматизированного управления памятью. В ходе разработки программ используется ограниченное адресное пространство, контроль над заполнением которого может лечь на плечи самих разработчиков или может быть делегирован средствам, которые автоматизируют этот процесс. В результате анализа принципов работы сборщика мусора на примере Java Hotspot были выявлены положительные и отрицательные черты, свойственные автоматизации процесса управления памятью.

Annotation: This article is devoted to an overview of the advantages and disadvantages inherent in automated memory management mechanisms. During the development of programs, a limited address space is used, control over the filling of which can fall on the shoulders of the developers themselves or can be delegated to tools that automate this process. As a result of the analysis of the principles of the garbage collector on the example of Java Hotspot, positive and negative features inherent in the automation of the memory management process were identified.

К*лючевые слова*: информатика, сборщик мусора, джава, программирование, управление памятью.

***K*ey words**: computer science, garbage collector, java, programming, memory management.

Разработка программного обеспечения является отдельным сектором современной промышленности, однако, как и любое другое производство, оно подвержено сдерживающим факторам, которые не допускают бесконтрольного развития. Как строители ограничены ресурсами при возведении зданий, так и программисты ограничены вычислительными мощностями, которые им доступны. В определенные моменты отсутствие должных вычислительных мощностей приводило к остановке развития целой отрасли, как, к примеру, случилось с машинным обучением в 80-е годы [1].

Основными ограничениями с точки зрения разработки классических программ являются [2]:

- Частота процессора, отвечающая за скорость проведения операций. Чем она выше, тем быстрее будет исполняться программа.
- Память, выделяемая компьютером для работы программы. В ходе разработки программисты занимают память сущностями, которые могут быть различны в зависимости от парадигмы, в которой ведется разработка. Однако и в объектно-ориентированном программировании, и в функциональном, память будет использоваться и заниматься для записи различных значений.

В этой статье будет рассмотрен именно процесс управления памятью и способы его оптимизации. Память компьютера ограничена, а программа использует её для хранения данных, необходимых для ее работы. Однако в большинстве случаев нет необходимости хранить всё. При исполнении процессов многие данные теряют актуальность и больше не будут востребованы и задействованы, то есть от них можно избавиться без угрозы нарушения исполнения. В программировании существует два основных формата контроля памяти: ручной (представлен в низкоуровневых языках, к примеру, C, Rust) и автоматизированный (представлен в высокоуровневых языках, к примеру, Java, C#, python, JavaScript).

Исходя из названия, в ручном управлении памятью программист должен самостоятельно прослеживать жизненный цикл создаваемых им объектов. Однако в ходе разработки профессионалам необходимо решать огромное количество проблем, начиная с задач высокого уровня абстракции по типу взаимодействия разрабатываемого кода со всей системой, заканчивая задачами низкого уровня абстракции по типу реализации необходимого функционала.

Чтобы снять с программистов необходимость ручного управления памятью, были созданы специализированные средства, автоматизирующие этот процесс, они были названы сборщиками мусора. Сборщик мусора - это форма автоматического управления памятью, который отслеживает, удаляет неиспользуемые объекты и реорганизовывает оставшиеся [3].

Сборщики мусора имеют различные реализации в зависимости от виртуальной машины, на которой будет выполняться программа. Однако большая часть из них работает по схожим принципам, которые будут рассмотрены на примере HotSpot Garbage Collector, реализованном в языке Java [4].

Прежде всего необходимо разобраться в структуре устройства памяти Java программ: она состоит из native memory (вся доступная память), heap (куча), metaspace(метаданные классов и статические переменные), stack(стек вызовов методов), codecache (кэшированный код).

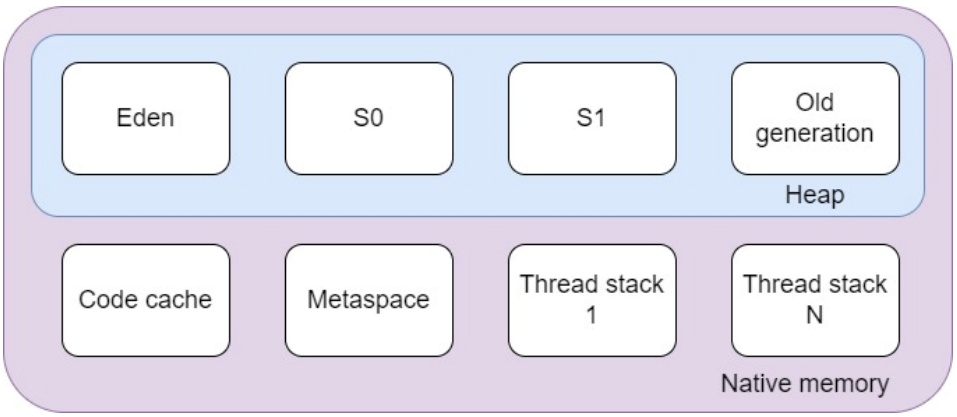

Рисунок 1 – Структура памяти

Как уже установилось ранее, сборка мусора - процесс удаления ненужных объектов из памяти, определение, в какой момент является бесполезным, происходит по следующему принципу: если в приложении он больше не будет использован, иными словами, когда на него больше нет ссылок.

Алгоритм сборки мусора, который реализован в Hotspot JVM, основывается на алгоритме пометок, состоящим из 3 этапов [5]:

- Маркировка: разметка всех живых объектов, для проведения этого этапа требуется приостановка приложения;
- Очистка: удаление немаркированных объектов:
- Компоновка: размещение оставшихся объектов в непрерывно друг за другом в памяти.

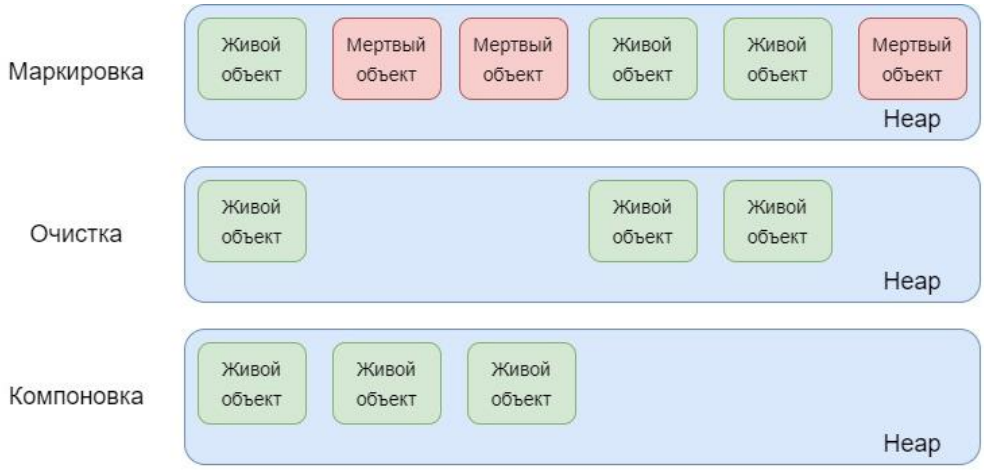

Рисунок 2 – Алгоритм разметки

Для любого программиста является очевидным факт того, что не все объекты имеют одинаковый жизненный цикл: некоторые создаются для отработки в рамках локальной области видимости, а некоторые задействованы

на протяжении всей работы программы. Исходя из того, что процесс сборки мусора требует дополнительных вычислительных затрат, можно понять необходимость оптимизации алгоритма, для этого была введена система поколений.

Механизм поколений представлен в виде 2 абстракций и четырех областей памяти в куче:

1. Молодое поколение включает в себя 3 области: eden, s1, s2, они служат для хранения новых объектов и переживших определенное количество сборок мусора соответственно.

2. Старое поколение используется для хранения объектов, переживших множество сборок мусора.

С учетом внедрения системы поколений алгоритм принимает следующий вид [6]:

1. Новые объекты помещаются в eden.

2. При заполнении eden запускается малая сборка мусора, которая удаляет и компонует объекты только в области памяти для новых поколений.

3. Объекты, выжившие после запуска малой сборки мусора, перемещаются в s0.

4. В ходе работы приложения eden будет время от времени заполняться новыми объектами, в результате чего будет запущен алгоритм малой сборки в областях eden и s0. Объекты, выжившие в результате этой сборки, перемещаются в область s1 с увеличением возраста.

5. При следующем запуске малой сборки процесс повторится, но с изменением области сохранения объектов: теперь они будут сохранены в s0 с увеличением возраста, а области eden, s1 будут очищены.

6. Объекты будут итерационно перемещаться между областями s0, s1 до тех пор, пока не достигнут определенного возраста или пока эти области не будут заполнены, после этого объекты будут перемещены в область old.

7. При необходимости будет запущен алгоритм большой сборки мусора, который выполняется для старого поколения.

На основе вышеизложенной информации о принципах работы сборщика мусора, можно выявить следующие преимущества технологии:

- Освобождение программиста от ручного освобождения памятью;
- Исключение попыток очистить пустое адресное пространство;
- Исключение всевозможных утечек памяти;
- Улучшение читаемости кода за счет исключения низкоуровневых команд;
- Повышение скорости разработки.

Однако за полностью автоматизированное управление памятью необходимо расплачиваться, работа сборщика мусора также требует вычислительных ресурсов, из чего вытекают следующие недостатки:

- Так как жизненный цикл объекта не подконтролен полностью программисту, то он не является строго определенным, что влечет за собой хранение в памяти тех объектов, которые уже бесполезны с точки зрения логики исполнения программы;
- В сравнении с программой с идеально настроенным жизненным циклом всех объектов, программы, использующие сборщик мусора, также используют в 5 раз больше памяти для исполнения и успешной работы [7].
- Для запуска большой сборки мусора требуется остановка приложения, а это может быть недопустимым для систем реального времени.

В результате анализа принципов работы сборщика мусора на примере Java Hotspot, были выявлены преимущества и недостатки, присущие технологии. Основной положительной чертой является снятие ответственности над контролем памяти для программиста, работающего над реализацией системы. В результате освобождаются ресурсы для выполнения более высокоуровневых задач проектирования, упрощается и ускоряется разработка программ. Однако необходимо иметь в виду, что сборщик мусора требует вычислительные ресурсы для работы и в некоторых случаях использование языков, поддерживающих его, является недопустимым: так, к примеру, компания “Apple” отказалась от

использования сборщиков мусора в операционной системе “iOS” именно по причине их ресурсозатратности [8].

Необходимо ли использовать эту технологию всегда или от нее следует отказаться? Ответ на этот вопрос индивидуален и полностью зависит от спецификации и требований, предъявляемым системе: если необходимо разработать приложение, которое будет максимально быстрым и оптимизированным и при этом заказчик готов потратить большие временные и денежные ресурсы, а сама среда исполнения ограничена в вычислительных мощностях, тогда следует выбрать низкоуровневые языки с ручным контролем памяти, если же необходимо в короткие сроки разработать приложение, которое не должно быть бесперебойным и максимально быстрым, тогда следует использовать языки высокого уровня, включающие в себя технологию сборки мусора.

СПИСОК ИСТОЧНИКОВ

1. История машинного обучения // Machine learning URL: http://www.machinelearning.ru/wiki/images/6/63/Voron2016ml-history.pdf (дата обращения: 14.11.2022).
2. The Fundamental Physical Limits of Computation // Scientific American URL: https://www.scientificamerican.com/article/the-fundamental-physical-limits-of-computation/ (дата обращения: 14.11.2022).
3. Dynamic storage allocation: A survey and critical review. Memory Management. Lecture Notes in Computer Science // Wilson P. R., Johnstone, M. S., Neely M. – Vol. 1 – CiteSeerX, 1995 – 986 с.
4. Java HotSpot Garbage Collection // Oracle URL: https://www.oracle.com/java/technologies/javase/hotspot-garbage-collection.html (дата обращения: 14.11.2022).
5. Dr. Edward Lavieri Mastering Java 11. - Second Edition - Packt Publishing, 2018. - 462 с.

6. HotSpot Virtual Machine Garbage Collection Tuning Guide // Oracle URL: https://docs.oracle.com/en/java/javase/16/gctuning/hotspot-virtual-machine-garbage-collection-tuning-guide.pdf (дата обращения: 14.11.2022).
7. Quantifying the Performance of Garbage Collection vs. Explicit Memory Management // Hertz, Matthew; Berger, Emery D. – OOPSLA – 2015 – 313 с.
8. Developers Tools Kickoff // Apple URL: https://docs.huihoo.com/apple/wwdc/2011/session_300__developer_tools_kickoff.pdf (дата обращения: 14.11.2022).

LIST OF SOURCES

1. History of machine learning // Machine learning URL: http://www.machinelearning.ru/wiki/images/6/63/Voron2016ml-history.pdf (date of application: 11/14/2022).
2. The Fundamental Physical Limits of Computation // Scientific American URL: https://www.scientificamerican.com/article/the-fundamental-physical-limits-of-computation / (date of request: 14.11.2022).
3. Dynamic storage allocation: A survey and critical review. Memory Management. Lecture Notes in Computer Science // Wilson P. R., Johnstone, M. S., Neely M. – Vol. 1 – CiteSeerX, 1995 – 986 c.
4. Java HotSpot Garbage Collection // Oracle URL: https://www.oracle.com/java/technologies/javase/hotspot-garbage-collection.html (date of application: 11/14/2022).
5. Dr. Edward Lavieri Mastering Java 11. - Second Edition - Packt Publishing, 2018. - 462 p.
6. HotSpot Virtual Machine Garbage Collection Tuning Guide // Oracle URL: https://docs.oracle.com/en/java/javase/16/gctuning/hotspot-virtual-machine-garbage-collection-tuning-guide.pdf (date of application: 14.11.2022).
7. Quantifying the Performance of Garbage Collection vs. Explicit Memory Management // Hertz, Matthew; Berger, Emery D. – OOPSLA – 2015 – 313 p .

8. Developers Tools Kickoff // Apple URL: https://docs.huihoo.com/apple/wwdc/2011/session_300__developer_tools_kickoff.pdf (date of application: 14.11.2022).

Для цитирования: Аникин Д.А. ПРЕИМУЩЕСТВА АВТОМАТИЗИРОВАННОГО УПРАВЛЕНИЯ ПАМЯТЬЮ НА ПРИМЕРЕ JAVA HOTSPOT // Научный сетевой журнал «Столыпинский вестник» №9/2022