Сборка мусора

Java Collections
4 уровень, 3 лекция

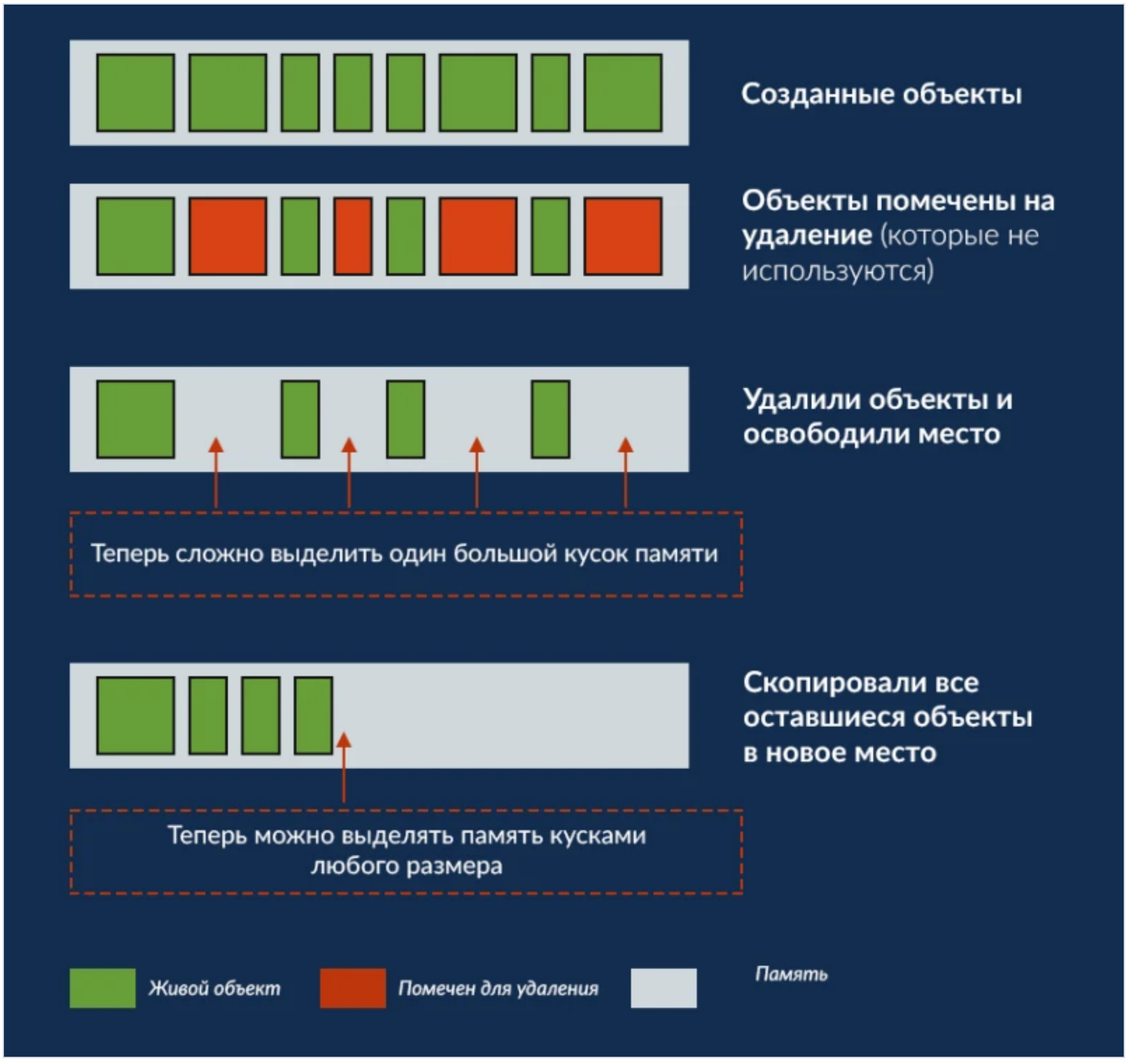
ОТКРЫТА

— Привет! Снова решила устроить тебе небольшую лекцию про сборку мусора.

Как ты уже знаешь, Java-машина сама отслеживает ситуации, когда объект становится ненужным и удаляет его.

— Ага. Вы с Ришей раньше мне рассказывали об этом, нюансов я не помню.

— Ок. Тогда повторим.



Как только объект создается, Java выделяет ему память. А за востребованностью объекта следит с помощью переменных-ссылок. Объект может быть удален при «сборке мусора» — процедуре очистки памяти, если не остается переменных, которые ссылаются на этот объект.

— А расскажи немного о сборщике мусора, что это такое и как он работает.

— Ок. Раньше сборка мусора происходила в главном потоке/нити. Раз в 5 минут, или чаще. Если наступал случай, когда не хватало свободной памяти, Java-машина приостанавливала работу всех нитей и удаляла неиспользуемые объекты.

Но сейчас от этого подхода отказались. Сборщик Мусора нового поколения работает незаметно и в отдельном потоке. Такую

— Ясно. А как именно определяется – нужно удалять объект или нет.

— Просто считать количество ссылок на объект не очень эффективно – ведь могут быть объекты, которые ссылаются друг на друга, но больше на них не ссылается никто.

Поэтому в Java применяется другой подход. Java делит объекты на 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.

— Ок. С этим вроде ясно.

А как происходит сама уборка мусора – удаление ненужных объектов?

— Тут все просто. В Java память условно разделена на две части, и когда приходит время сборки мусора, все живые (достижимые) объекты копируются в другую часть памяти, а старая память вся освобождается.

— Интересный подход. И не надо считать ссылки – скопировал все достижимые объекты, а все остальные – мусор.

— Там все немного сложнее. Программисты Java выяснили, что объекты обычно делятся на две категории – долгоживущие (которые существуют все время работы программы) и маложивущие (нужны в методах и для выполнения «локальных» операций).

Хранить долгоживущие отдельно от маложивущих гораздо эффективнее. Для этого надо было придумать механизм определения долгожительства объекта.

Поэтому они разделили всю память на «поколения». Есть объекты первого поколения, есть объекты второго поколения и т.д. Каждый раз после очистки памяти счетчик поколений увеличивается на 1. Если какие-то объекты существуют много поколений, то их записывали в долгожители.

Сборщик Мусора очень — сложная и эффективная составляющая Java. Многие его части работают эвристически – на основе алгоритмов-догадок. Поэтому он часто «не слушается» пользователя.

— В смысле?

— У Java есть объект GC (Garbage Collector – Сборщик Мусора), который можно вызвать с помощью метода System.gc().

Также можно принудительно инициировать вызов finalize-методов удаляемых объектов, посредством System.runFinalization(). Но дело в том, что по документации Java, это не гарантирует ни начало сборки мусора, ни вызов методов finalize(). Garbage Collector сам решает, что и когда ему вызывать.

— Ничего себе! Буду знать.

— Но и это еще не все. Как ты знаешь, в Java одни объекты ссылаются на другие, и именно с помощью этой сети ссылок определяется – стоит удалять объект или нет.

Так вот, в Java есть специальные ссылки, которые позволяют влиять на этот процесс. Для них есть специальные классы-обертки. Вот они:

SoftReference – мягкая ссылка.

WeakReference – слабая ссылка.

PhantomReference – призрачная ссылка.

— М-да. Чем-то напоминает внутренние классы, вложенные классы, внутренние анонимные классы, локальные классы. Названия разные, но по ним совсем не понятно за что они отвечают.

— Вот ты, Амиго, и стал программистом. Теперь ты возмущаешься по поводу названий классов – дескать, недостаточно информативны, и нельзя по одному названию(!) определить, что этот класс делает, как и зачем.

— Ого. А я и сам не заметил. Но это же так очевидно.

Ладно. Соловья баснями не кормят. Давай я тебе расскажу про SoftReference – мягкие ссылки.

Эти ссылки были специально придуманы для кэширования, хотя их можно использовать и для других целей – все на усмотрение программиста.

Пример такой ссылки:

Пример

```
1  //создание объекта Cat
2  Cat cat = new Cat();
3
4  //создание мягкой ссылки на объект Cat
5  SoftReference<Cat> catRef = new SoftReference<Cat>(cat);
6
7  //теперь на объект ссылается только мягкая ссылка catRef.
8  cat = null;
9
10 //теперь на объект ссылается еще и обычная переменная cat
11 cat = catRef.get();
12
13 //очищаем мягкую ссылку
14 catRef.clear();
```

Если на объект существуют только мягкие ссылки, то он продолжает жить и называется «мягкодостижимым».

Но! 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.

Предположим, что программа-клиент часто запрашивает у программы-сервера различные данные. Тогда программа сервер может некоторые из них кэшировать, воспользовавшись для этого SoftReference. Если объекты, удерживаемые от смерти мягкими ссылками, будет занимать большую часть памяти, то сборщик мусора просто их поудаляет и все. Красота!

— Ага. Мне самому понравилось.

— Ну и маленькое дополнение: у класса SoftReference есть два метода. Метод get() возвращает объект, на который ссылается SoftReference. Если объект был удален сборщиком мусора, внезапно(!) метод **get**() начнет отдавать null.

Так же пользователь может сам очистить SoftReference, вызвав метод clear(). При этом слабая ссылка внутри объекта SoftReference будет уничтожена.

На этом пока все.

— Спасибо за интересный рассказ, Элли. Действительно было очень интересно.

‹ Предыдущая лекция

Следующая лекция ›

В общем сборщик мусора работает мудрено, нюансов я не запомню. Хотите управлять памятью понятно и эффективно - велком в C++, будете ручками там выделять и удалять память на каждый объект.

Ответить 0

LuneFox инженер по сопровождению в **BIFIT** EXPERT 18 января, 20:03

То есть, объект, на который ссылается только мягкая ссылка, "нужен, но не очень"?

-- Эй программист, тебе ещё нужен этот объект?
-- Ну... не знаю... наверное да... а хотя погоди, не... а вдруг понадобится? Маловеоятно, конечно, но всякое может быть, ну в общем, если место есть, поставь куда нибудь, ну а если нет, ладно, выкидывай.

Ответить +5

Роман Кончалов Уровень 28, Россия EXPERT 31 марта, 20:48

Да, кеширование так и работает. Сейчас данные не нужны, но потом могут пригодиться, хотя в крайнем случае их можно и заново загрузить из другого места, если кончилось место в оперативке и данные были удалены.
Поэтому многие ноют, что Chrome жрёт слишком много памяти, хотя он жрёт ровно столько сколько свободно (с некоторым ограничением: 60-80% от свободной памяти). И когда мы открываем много вкладок они естественно кешируются сначала в оперативную память, а потом на жёсткий диск. И чем больше у нас оперативная память, тем больше хром постарается оставить в ней, чтобы не обращаться к медленному жёсткому диску или изнашиваемому SSD.

Ответить +2

Игорь Full Stack Developer в **IgorApplications** 26 июля 2021, 12:17

Динамический массив на си++
```
int *arr = new int[size];
delete []arr;
```
Выделяем байтитки для динамического массива в си
```
int *arr = malloc(sizeof(int) * size);
free(arr);
```

Ответить 0

Олег Д Уровень 40, Москва, Россия 15 июля 2021, 18:44

мягкодостижимый объект)) плюсуй, если бывшую свою вспомнил))

Ответить +4

Denis Уровень 40, Москва, Russia 9 октября 2021, 12:06

Ответить 0

Андрій Мовчан Уровень 48, Львов 31 октября 2021, 01:14

Годно😂

Ответить 0

LuneFox инженер по сопровождению в **BIFIT** EXPERT 18 января, 17:27

Жёны декабристов тоже считали своих мужей мягкодостижимыми, потому смогли достигнуть их через ссылку?

Ответить +10

Maks Panteleev Java Developer в **Bell Integrator** 13 июля 2021, 10:15

Амиго стал программистом, а ты - нет.

Ответить +19

Андрій Мовчан Уровень 48, Львов 31 октября 2021, 01:14

Даже как то обидно(

Ответить +1

Александр Черенков Уровень 37, Бердск, Россия 17 апреля 2021, 09:56

Статья про ссылки на JavaRush

Ответить +2

Dmitry Gidlevsky Уровень 35, Киев, Украина 24 марта 2021, 02:05

Лаконично и просто)

Strong Reference - GC никогда не удалит объект

GC - garbage collector

Ответить +3

LuneFox инженер по сопровождению в **BIFIT** EXPERT 18 января, 17:28

Phantom reference - тебя мучают мысли о фантомном объекте, которого нет :)

Ответить +1

Luicich Уровень 39, Минск 13 февраля 2021, 22:14

ой муть пошла...

Ответить +5

Иван Уровень 41, Москва 11 декабря 2020, 16:04

Не совсем ясно зачем делать мягкие ссылки, в каких именно случаях их нужно использовать, что делать если всё было удалено,
и как правильно описывать и работать в коде по мягким ссылкам.

Ответить +6

Pig Man Главная свинья в **Свинарнике** 7 февраля 2021, 17:06

«Не совсем ясно» - это слоган данного курса

Ответить +26

Роман Кончалов Уровень 28, Россия EXPERT 31 марта, 20:53

Мягкие ссылки нужны для тех объектов, которые необходимо кешировать. Кешировать необходимо такие объекты, которые долго инициализируются, но их нужно получать быстро. То есть мы можем закешировать объект после первого вызова или заранее, и хранить его в кеше, пока позволяет память. Естественно, что кешировать можно только те объекты, которые можно получить заново не из кеша в случае их удаления из кеша.

Ответить +1

Artem K. Уровень 30, Москва 15 ноября 2020, 20:17

Блин, не хватало этой лекции, когда дерево решали... Всех потомков удаляемого элемента можно было и не перебирать и удалять, так как они "не достижимые".
Ну на будущее будем знать)

Ответить +4

Показать еще комментарии

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store