

Professor Hans Noodles

41 уровень

22.05.2019 12321 20

Особенности PhantomReference

Статья из группы Java Developer

43181 участник

Вы в группе

Привет!

На сегодняшнем занятии мы подробно поговорим о «**фантомных ссылках**» (**PhantomReference**) в Java. Что это за ссылки такие, почему называются «фантомными» и как ими пользоваться?



Как ты помнишь, в Java есть 4 вида ссылок:

- 1. **StrongReference** (обычные ссылки, которые мы создаем при создании объекта):

```
Cat cat = new Cat()
```

cat в этом примере — Strong-ссылка.

- 2. **SoftReference** (мягкая ссылка). У нас была [лекция](#) про эти ссылки.
- 3. **WeakReference** (слабая ссылка). Про них тоже была лекция, [вот](#).
- 4. **PhantomReference** (фантомная ссылка).

Объекты трех последних видов типизированные (например, `SoftReference<Integer>`, `WeakReference<MyClass>`).

НАЧАТЬ ОБУЧЕНИЕ

Наиболее важные методы при работе с этими классами:

- `get()` — возвращает объект, на который ссылается эта ссылка;
- `clear()` — удаляет ссылку на объект.

Эти методы ты помнишь по лекциям о `SoftReference` и `WeakReference`. Важно помнить, что они работают по-разному с разными видами ссылок.

Мы сегодня не будем подробно рассматривать первые три типа, а поговорим о фантомных ссылках. Остальные виды ссылок мы тоже затронем, но только в той части, где будем говорить, чем фантомные ссылки от них отличаются.

Поехали! :)

Начнем с того, зачем нам вообще нужны фантомные ссылки.

Как ты знаешь, освобождением памяти от ненужных объектов Java занимается **сборщик мусора** (Garbage Collector или gc).

Сборщик удаляет объект в два «прохода».

В первый проход он только смотрит на объекты, и, если надо, помечает его как «ненужный, подлежащий удалению». Если у этого объекта был переопределен метод `finalize()`, он вызывается. Или не вызывается — как повезет. Ты наверняка помнишь, что `finalize()` — штука непостоянная :)

Во второй проход сборщика объект удаляется, и память освобождается.

Такое непредсказуемое поведение сборщика мусора создает для нас ряд проблем.

Мы не знаем когда именно начнется работа сборщика мусора. Мы не знаем будет ли вызван метод `finalize()`. Плюс ко всему, во время работы `finalize()` может быть создана strong-ссылка на объект, и тогда он вообще не будет удален. В системах, требовательных к объему свободной памяти, это может легко привести к `OutOfMemoryError`.

Все это подталкивает нас к использованию **фантомных ссылок**.

Дело в том, что это меняет поведение сборщика мусора. Если на объект остались только фантомные ссылки, то у него:

- вызывается метод `finalize()` (если он переопределен);
- если после работы `finalize()` ничего не изменилось и объект все еще может быть удален, фантомная ссылка на объект помещается в специальную очередь — `ReferenceQueue`.

Самое важное, что нужно понимать при работе с фантомными ссылками, — **объект не удаляется из памяти до тех пор, пока его фантомная ссылка находится в этой очереди**.

Он будет удален только после того, как у фантомной ссылки будет вызван метод `clear()`.

Давай рассмотрим пример. Для начала создадим тестовый класс, который будет хранить в себе какие-то данные.

```
1 public class TestClass {
2     private StringBuffer data;
3     public TestClass() {
4         this.data = new StringBuffer();
5         for (long i = 0; i < 50000000; i++) {
6             this.data.append('x');
7         }
8     }
9 }
```

```
10         protected void finalize() {
11             System.out.println("У объекта TestClass вызван метод finalize!!!");
12         }
13     }
```

Мы специально как следует «загружаем» объекты данными при создании (добавляем в каждый объект по 50 миллионов символов «х»), чтобы занять побольше памяти.

Кроме того, мы специально переопределяем метод `finalize()`, чтобы увидеть, что он сработал.

Далее нам понадобится класс, который будет наследоваться от `PhantomReference`. Зачем нам нужен такой класс?

Все просто. Так мы сможем добавить дополнительную логику к методу `clear()`, чтобы увидеть, что очистка фантомной ссылки действительно произошла (а значит, объект удален).

```
1  import java.lang.ref.PhantomReference;
2  import java.lang.ref.ReferenceQueue;
3
4  public class MyPhantomReference<TestClass> extends PhantomReference<TestClass> {
5
6      public MyPhantomReference(TestClass obj, ReferenceQueue<TestClass> queue) {
7
8          super(obj, queue);
9
10         Thread thread = new QueueReadingThread<TestClass>(queue);
11
12         thread.start();
13     }
14
15     public void cleanup() {
16         System.out.println("Очистка фантомной ссылки! Удаление объекта из памяти!");
17         clear();
18     }
19 }
```

Далее, нам понадобится отдельный поток, который будет ждать, пока сборщик мусора сделает свое дело, и в нашей очереди `ReferenceQueue` появятся фантомные ссылки. Как только такая ссылка попадет в очередь, у нее будет вызван метод `cleanup()`:

```
1  import java.lang.ref.Reference;
2  import java.lang.ref.ReferenceQueue;
3
4  public class QueueReadingThread<TestClass> extends Thread {
5
6      private ReferenceQueue<TestClass> referenceQueue;
7
8      public QueueReadingThread(ReferenceQueue<TestClass> referenceQueue) {
9          this.referenceQueue = referenceQueue;
10     }
11
12     @Override
13     public void run() {
14         while (true) {
15             Reference<TestClass> ref = referenceQueue.poll();
16             if (ref != null) {
17                 ref.clear();
18             }
19         }
20     }
21 }
```

```
16      Reference ref = null;
17
18      //ждем, пока в очереди появятся ссылки
19      while ((ref = referenceQueue.poll()) == null) {
20
21          try {
22              Thread.sleep(50);
23          }
24
25          catch (InterruptedException e) {
26              throw new RuntimeException("Поток " + getName() + " был прерван!");
27          }
28      }
29
30      //как только в очереди появилась фантомная ссылка - очистить ее
31      ((MyPhantomReference) ref).cleanup();
32  }
33 }
```

И, наконец, нам понадобится метод `main()`: вынесем его в отдельный класс `Main`.

В нем мы создадим объект `TestClass`, фантомную ссылку на него и очередь для фантомных ссылок. После этого мы вызовем сборщик мусора и посмотрим, что будет :)

```
1  import java.lang.ref.*;
2
3  public class Main {
4
5      public static void main(String[] args) throws InterruptedException {
6          Thread.sleep(10000);
7
8          ReferenceQueue<TestClass> queue = new ReferenceQueue<>();
9          Reference ref = new MyPhantomReference<>(new TestClass(), queue);
10
11          System.out.println("ref = " + ref);
12
13          Thread.sleep(5000);
14
15          System.out.println("Вызывается сборка мусора!");
16
17          System.gc();
18          Thread.sleep(300);
19
20          System.out.println("ref = " + ref);
21
22          Thread.sleep(5000);
23
24          System.out.println("Вызывается сборка мусора!");
25
26          System.gc();
27      }
28  }
```

ref = MyPhantomReference@4554617с
Поток, отслеживающий очередь, стартовал!
Вызывается сборка мусора!
У объекта TestClass вызван метод finalize!!!
ref = MyPhantomReference@4554617с
Вызывается сборка мусора!
Очистка фантомной ссылки! Удаление объекта из памяти!

Что же мы здесь видим?

Все произошло, как мы и планировали!

У нашего класса объекта был переопределен метод `finalize()`, и он был вызван во время работы сборщика.

Далее, фантомная ссылка была помещена в очередь `ReferenceQueue`. Там у нее был вызван метод `clear()` (из которого мы сделали `cleanup()`), чтобы добавить вывод в консоль).

В итоге объект был удален из памяти.

Теперь ты видишь, как именно это работает :)

Конечно, тебе не нужно зазубривать наизусть всю связанную с фантомными ссылками теорию. Но будет хорошо, если ты будешь помнить хотя бы главные моменты.

Во-первых, это самые слабые ссылки из всех. Они вступают в работу только когда на объект не осталось никаких других ссылок.

Список ссылок, которые мы привели выше, идет по «убыванию силы»:

`StrongReference` -> `SoftReference` -> `WeakReference` -> `PhantomReference`

Фантомная ссылка вступит в бой только когда на наш объект не будет ни Strong, ни Soft, ни Weak ссылок :)

Во-вторых, метод `get()` для фантомной ссылки всегда возвращает `null`.

Вот простой пример, где мы создаем три разных типа ссылок для трех разных видов автомобилей:

```
1  import java.lang.ref.PhantomReference;
2  import java.lang.ref.ReferenceQueue;
3  import java.lang.ref.SoftReference;
4  import java.lang.ref.WeakReference;
5
6  public class Main {
7
8      public static void main(String[] args) {
9
10         Sedan sedan = new Sedan();
11         HybridAuto hybrid = new HybridAuto();
12         F1Car f1car = new F1Car();
13
14         SoftReference<Sedan> softReference = new SoftReference<>(sedan);
15         System.out.println(softReference.get());
16
17         WeakReference<HybridAuto> weakReference = new WeakReference<>(hybrid);
```

НАЧАТЬ ОБУЧЕНИЕ

```
19
20         ReferenceQueue<F1Car> referenceQueue = new ReferenceQueue<>();
21
22         PhantomReference<F1Car> phantomReference = new PhantomReference<>(f1car, referenceQueue);
23         System.out.println(phantomReference.get());
24
25     }
26 }
```

Вывод в консоль:

Sedan@4554617c
HybridAuto@74a14482
null

Метод `get()` вернул вполне нормальные объекты для мягкой ссылки и слабой ссылки, но вернул `null` для фантомной.

В-третьих, основная область использование фантомных ссылок — сложные процедуры удаления объектов из памяти.

Вот и все! :) На этом наше сегодняшнее занятие окончено.

Но на одной теории далеко не уедешь, поэтому пора возвращаться к решению задач! :)

+65

Комментарии (20)

популярные новые старые

JavaCoder

Введите текст комментария

Дмитрий Уровень 36 30 июля 2021, 17:45

<https://otus.ru/nest/post/304/>
понятно и просто о фантоме

Ответить +6

Николай Т. Уровень 40, Рязань, Россия 10 июля 2021, 13:10

Я почувствовал какие-то присутствие у себя за спиной, обернулся, но мои глаза вернули null :)

Ответить +11

Valua Sinicyn Уровень 41, Харьков, Украина 11 февраля 2021, 09:46

Ссылки на человеческом: [Тап](#)

Примеры фантомных ссылок: [Тап](#)

Ответить +4

Андрей Уровень 41, Самара 25 января 2021, 17:25

<https://www.programcreek.com/java-api-examples/?api=java.lang.ref.PhantomReference>
если кому поможет, примеры использования PhantomReference из реальных проектов

Ответить +1

НАЧАТЬ ОБУЧЕНИЕ

Что-то я запутался.
Сборщик мусора - Garbage Collector или Garbage Collection?
В разных источниках по разному почему-то.

Ответить

Mike

Уровень 35, Москва, Россия

14 октября 2020, 15:28

...

1й вариант.

Ответить

funbiscuit

Уровень 41, Россия

15 января 2021, 13:14

...

Garbage Collector - сам сборщик мусора.
Garbage Collection - процесс сборки мусора (сборка мусора).

Ответить

+1

Евгений Буш

Программист в Компания Nordside

EXPERT

30 марта 2020, 20:20

...

экспериментировал на этом примере, после .clear() память, занимаемая программой не уменьшалась, значит объект так и не уничтожался. Какой- то косяк. [эксперимент с фантомом](#)

Ответить

0

Leftover

Уровень 39, Москва, Россия

14 января 2020, 16:38

...

>основная область использование фантомных ссылок — сложные процедуры удаления объектов из памяти.
Типа вам не понять....

Ответить

+8

Андрей

Уровень 41, Москва, Россия

15 декабря 2020, 13:42

...

Я так и понял, что пока мне не понадобится что-то непонятное - мне это не нужно

Ответить

+1

Александр

Уровень 1, Москва

14 декабря 2019, 18:08

...

Мне кажется, в тексте Main потеряли старт потока QueueReadingThread, отслеживающего очередь.

Ответить

0

Vitaly Khan

Java Developer в Onollo

MASTER

17 декабря 2019, 05:39

...

нет, не потеряли.
в 9-й строке класса Main создается объект MyPhantomReference. вот в его конструкторе и стартует поток отслеживающий очередь.

Ответить

+2

Oleg Martynov

Уровень 41, Санкт-Петербург, Россия

7 декабря 2019, 02:36

...

Почему же PhantomReference считается слабее WeakReference, если у обеих ссылок время жизни объекта от сборки до сборки? К тому же объект, на который смотрит WeakReference, убивается коллектором сразу, а объект PhantomReference живет в памяти, пока не вызван метод clear().

Ответить

+1

Vitaly Khan

Java Developer в Onollo

MASTER

17 декабря 2019, 05:43

...

наверно, потому что WeakReference способна возвращать вам объект, а PhantomReference - нет. т.е. рассуждать нужно с точки зрения доступности объекта. она потому и фантомная. вроде как от объекта что-то осталось (возможность выполнить какой-то метод после finalize), но самого объектам, можно считать, нет, т.к. он недоступен.

Ответить

+3

Александр

Уровень 41, Москва, Россия

EXPERT

5 октября 2019, 21:58

...

получается, фантомная ссылка хоть и самая слабая, но удерживает объект в памяти как сильная. собака на сене.

Ответить

+1

Показать еще комментарии

ОБУЧЕНИЕ

Курсы программирования

СООБЩЕСТВО

Пользователи

КОМПАНИЯ

О нас

НАЧАТЬ ОБУЧЕНИЕ



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

▼

