phantom Reference

Java Collections
4 уровень, 7 лекция

ОТКРЫТА

— Привет, Амиго!

— Привет, Риша!

— Ну, как день прошел?

— Отлично! Мне сегодня Билаабо рассказал про рекурсию, а Элли — про слабые и мягкие ссылки.

— А про призрачные ссылки рассказывала?

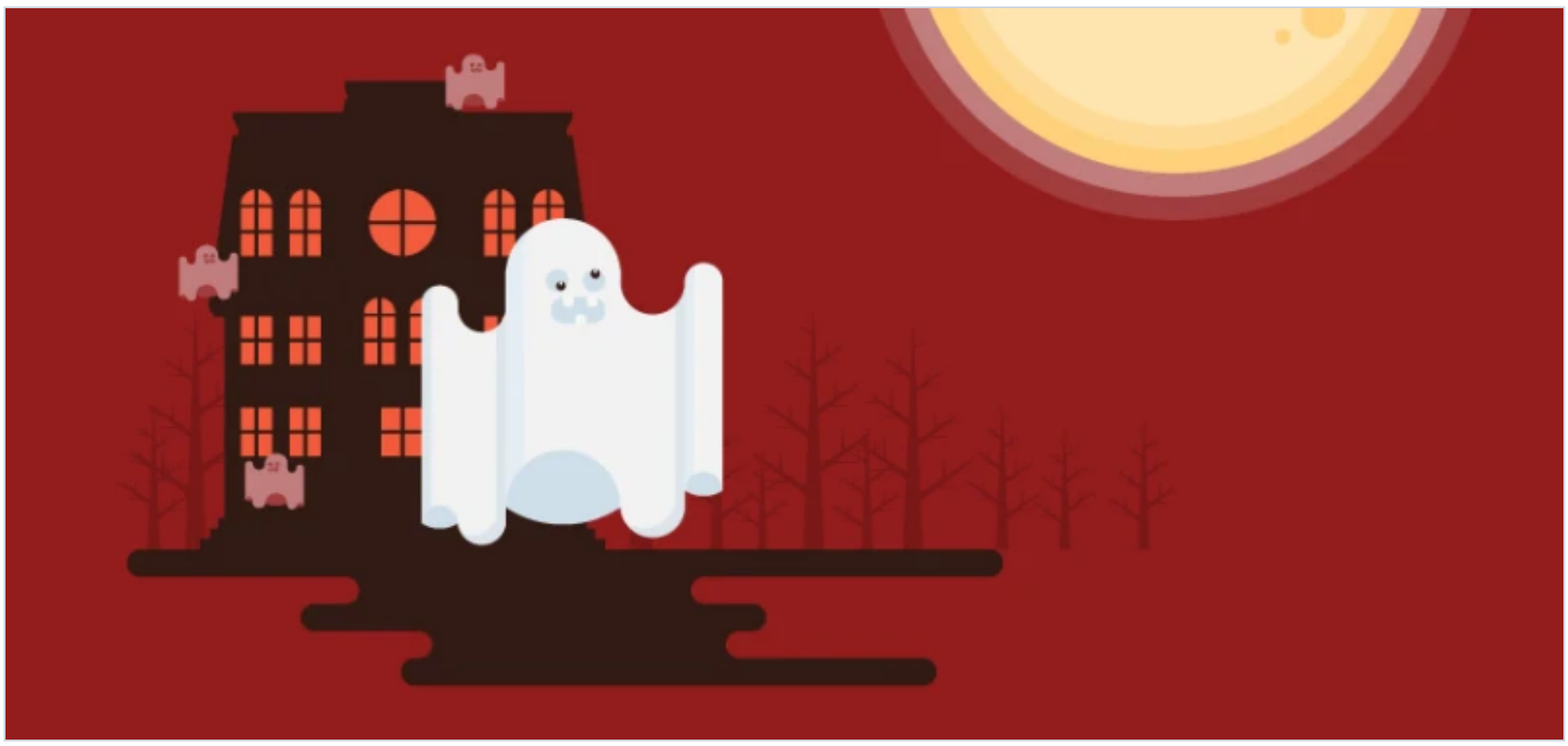
— Ты про PhantomReference? Упоминала, но не рассказывала подробно.

— Отлично, тогда надеюсь, ты не будешь против, если я заполню этот пробел.

— Конечно, я с удовольствием тебя послушаю, Риша!

— Отлично. Тогда я начну.

Призрачные (Phantom) ссылки – это самые слабые ссылки из всех. Только если на объект не остаётся никаких ссылок вообще, кроме призрачных, их механизм вступает в действие.



PhantomReference используется для сложной процедуры удаления объекта. Это может быть необходимо, когда объект что-то делает за границами Java-машины, например вызывает низкоуровневые функции ОС или пишет свое состояние в файл или еще что-нибудь очень важное.

Пример использования:

Пример создания призрачных ссылок

```
1  //специальная очередь для призрачных объектов
2  ReferenceQueue<Integer> queue = new ReferenceQueue<Integer>();
3
4  //список призрачных ссылок
```

НАЧАТЬ ОБУЧЕНИЕ

```
7  //создаем 10 объектов и добавляем их в список через призрачные ссылки
8  for ( int i = 0; i < 10; i++)
9  {
10  Integer x = new Integer(i);
11  list.add(new PhantomReference<Integer>(x, queue));
12 }
```

Еще раз обращаю внимание на последнюю строчку. В PhantomReference передается не только объект x, но и специальная очередь призрачных ссылок.

— А зачем нужна эта очередь?

— Вот сейчас и расскажу.

При уничтожении объекта, удерживаемого призрачной ссылкой, он уничтожается, но не удаляется из памяти! Вот такая вот загогулина, понимаешь.

— А это как?

— Тут довольно много нюансов, так что начну с самого простого.

Если на объект остаются только призрачные ссылки, то вот что его ждет:

Шаг 1. Во время ближайшей сборки мусора у объекта будет вызван метод finalize(). Но, если метод finalize() не был переопределен, этот шаг пропускается, а выполнится сразу шаг 2.

Шаг 2. Во время следующей сборки мусора, объект будет помещен в специальную очередь призрачных объектов, из которой будет удален, когда у PhantomReference вызовут метод clear().

— А кто его вызовет? Ведь объект-то как бы удален, разве нет?

— Тут дело в том, что фактически объект умер в нашем (Java) мире, но не исчез, а остался в нем призраком – на него хранится ссылка в очереди призрачных объектов. Та самая ReferenceQueue, ссылку на которую мы так заботливо передаем в конструктор PhantomReference.

— Т.е. эта ReferenceQueue — это как бы потусторонний мир?

— Скорее, как мир призраков.

И чтобы удалить объект-призрак, надо вызвать clear() у его призрачной ссылки.

Вот как можно продолжить предыдущий пример:

Пример создания призрачных ссылок

```
1  //специальная очередь для призрачных объектов
2  ReferenceQueue<Integer> queue = new ReferenceQueue<Integer>();
3
4  //список призрачных ссылок
5  ArrayList<PhantomReference<Integer>> list = new ArrayList<PhantomReference<Integer>>();
6
7  //создаем 10 объектов и добавляем их в список через призрачные ссылки
8  for ( int i = 0; i < 10; i++)
9  {
10  Integer x = new Integer(i);
11  list.add(new PhantomReference<Integer>(x, queue));
12 }
13
14 //взываем сборщик мусора, надеемся, что он нас послушается :)
```

```
System.gc();

//достаем из очереди все объекты
Reference<? extends Integer>referenceFromQueue;
while ((referenceFromQueue = queue.poll()) != null)
{
 //выводим объект на экран
 System.out.println(referenceFromQueue.get());
 //очищаем ссылку
 referenceFromQueue.clear();
}
```

— Что что-то тут происходит – это понятно. Даже почти понятно, что именно происходит.

Но как это использовать на практике?

— Вот тебе более адекватный пример:

Пример создания призрачных ссылок

```
//специальная очередь для призрачных объектов
ReferenceQueue<Integer> queue = new ReferenceQueue<Integer>();

//список призрачных ссылок
ArrayList<PhantomInteger> list = new ArrayList<PhantomInteger>();

//создаем 10 объектов и добавляем их в список через призрачные ссылки
for ( int i = 0; i < 10; i++)
{
 Integer x = new Integer(i);
 list.add(new PhantomInteger (x, queue));
}
```

Эта нить будет следить за призрачной очередью и удалять оттуда объекты

```
Thread referenceThread = new Thread()
{
 public void run()
 {
  while (true)
  {
   try
   {
    //получаем новый объект из очереди, если объекта нет - ждем!
    PhantomInteger ref = (PhantomInteger)queue.remove();
    //вызвваем у него метод close
    ref.close();
    ref.clear();
   }
   catch (Exception ex)
   {
    // пишем в лог ошибки
   }
  }
 }
```

```
referenceThread.setDaemon(true);
referenceThread.start();
```

Это класс, унаследованный от PhantomReference, у него есть метод close()

```
static class PhantomInteger extends PhantomReference<Integer>
{
 PhantomInteger(Integer referent, ReferenceQueue<? super Integer> queue)
 {
  super(referent, queue);
 }

 private void close()
 {
  System.out.println("Bad Integer totally destroyed!");
 }
}
```

Мы тут сделали три вещи.

Во-первых, мы создали класс PhantomInteger, который унаследовали от PhantomReference<Integer>.

Во-вторых, у этого класса есть специальный метод – close(), ради вызова которого как бы все это и затевается.

В третьих, мы объявили специальную нить — referenceThread. Она в цикле ждет, пока в очереди призраков не появится еще один объект. Как только он появляется, она удаляет его из очереди призраков, а затем вызывает у него метод close(). А затем метод clear(). И все – призрак может переходить в следующий лучший мир. В нашем он нас больше не побеспокоит.

— Как интересно, однако все вышло.

— Мы фактически отслеживаем очередь умирающих объектов, и потом для каждого можем вызвать специальный метод.

Но, учти, ты не можешь вызвать метод самого объекта. Ссылку на него получить нельзя! Метод get() у PhantomReference всегда возвращает null.

— Но ведь мы же наследуемся от PhantomReference!

— Даже внутри наследника PhantomReference, метод get() возвращает null.

— Тогда я просто сохраню ссылку на объект в конструкторе

— Ага. Но тогда эта ссылка будет StrongReference, и объект никогда не попадет в очередь призраков!

— Блин. Ладно, сдаюсь. Нельзя так нельзя.

— Вот и отлично. Надеюсь, ты вынесешь для себя что-то ценное из сегодняшнего урока.

— Да тут столько нового материала. А я думал, что уже все знаю. Спасибо тебе за урок, Риша.

— Пожалуйста. Все, иди отдыхай. Но не забудь, у нас вечером еще урок.

‹ Предыдущая лекция

Следующая лекция ›

Комментарии (83) популярные **новые** старые

JavaCoder

Введите текст комментария

Anonymous #2957882 Уровень 41, Днепр, Ukraine — 8 июля, 11:17

Шалом, Риша!

Ответить 0

Andrey Karelin Уровень 41, Sumy, Украина — 30 апреля, 00:53

Со ссылками Strong и Soft еще как-то более-менее понятно.
С Phantom - с очень большой натяжкой.
Для чего необходима Weak совсем не ясно. Если эти ссылки уничтожит ближайший сборщик мусора...который приходит сам по себе (по внутреннему механизму JVM), и КОГДА именно он это сделает совсем не прогнозируемо, то зачем нам оставлять такие ссылки?
Все равно, что вынести мусорное ведро в бак с нужной вещью, и надеяться, что она вам понадобится раньше, чем баки вывезет мусоровозка.

Ответить 0

Макс Дудин Уровень 39, Калининград, Россия — 26 февраля, 21:23

замут... по любому это "но учти ты не можешь вызвать метод самого объекта" где дальше будет "лежать" в качестве подводного камня....

Ответить +1

Greatsky future developer в **future developer** — 13 января, 17:34

Я думаю на основании это статьи можно написать сюжет для фильма матрицы, аля той когда Нео попал на станцию и не могу выйти пока не очень красивый персонаж позволил ему есть в вагон))) 😂

Ответить +1

Yarik Таксист в **Яндекс.Такси** — 5 декабря 2021, 07:54

Сложно воспринимать материал когда нет понимания для чего это использовать, ведь GC и так неплохо справляется со своими задачами.

Ответить +6

AlinaAlina Уровень 35, Санкт-Петербург — 1 декабря 2021, 17:54

"Вот тебе более адекватный пример", - сказал Риша.
Жаль только, что он ещё менее адекватный, и что у меня он почему-то ничего не выводит. Может кто-то что-то понял лучше, и может объяснить?

Ответить +6

Kes Чайник в **Банк** — 21 ноября 2021, 08:10

Мягкие ссылки на страже доступной памяти или как экономить память правильно

Ответить +3

Andrey Gordeev Уровень 36, Минск — 28 сентября 2021, 09:24

Шаг 1. Во время ближайшей сборки мусора у объекта будет вызван метод finalize(). Но, если метод finalize() не был переопределен, этот шаг пропускается, а выполнится сразу шаг 2.

Вопрос относительно этого высказывания: гарантированно ли вызывается finalize() у призрачных объектов, если переопределён?

Шаг 2. Во время следующей сборки мусора, объект будет помещен в специальную очередь призрачных объектов, из которой будет удален, когда у PhantomReference вызовут метод clear().

Вопрос: выполнение Шаг 1 или пропуск Шаг 1 считается за 1 цикл сборки мусора? А Шаг 2 считается 2 циклом? или при skipe Шага 1, Шаг 2 будет 1 циклом сборки?

Ответить +1

Anonymous #2489173 Уровень 35 — 22 сентября 2021, 10:02

"— Вот тебе более адекватный пример:"
Но ведь даже неясно практическое применение особо, что адеватного-то

Ответить +1

Задают вопрос jvm:
-Видишь ссылку в память?
-Нет.
-А она там есть...

Ответить

+1

Показать еще комментарии