关于Java中的WeakReference

BrightLoong (/u/c19e2c35c1ae) + 关注
1.4 2018.05.27 11:28* 字数 548 阅读 16564 评论 3 喜欢 18
(/u/c19e2c35c1ae)

阅读原文请访问我的博客BrightLoong's Blog (https://brightloong.github.io/2018/05/27/%E5%85%B3%E4%BA%8EJava%E4%B8%AD%E7%9A%84WeakReference/#more)

一. 简介

在看ThreadLocal源码的时候，其中嵌套类ThreadLocalMap中的Entry继承了WeakReferenc，为了能搞清楚ThreadLocal，只能先了解下了WeakReferenc(是的，很多时候为了搞清楚一个东西，不得不往上追好几层，先搞清楚其所依赖的东西。)
下面进入正题，WeakReference如字面意思，弱引用， 当一个对象仅仅被weak reference（弱引用）指向, 而没有任何其他strong reference（强引用）指向的时候, 如果这时GC运行, 那么这个对象就会被回收，不论当前的内存空间是否足够，这个对象都会被回收。

二. 认识WeakReference类

WeakReference继承Reference，其中只有两个构造函数：

```
public class WeakReference<T> extends Reference<T> {
    public WeakReference(T referent) {
        super(referent);
    }

    public WeakReference(T referent, ReferenceQueue<? super T> q) {
        super(referent, q);
    }

}
```

- WeakReference(T referent)：referent就是被弱引用的对象（注意区分弱引用对象和被弱引用的对应，弱引用对象是指WeakReference的实例或者其子类的实例），比如有一个Apple实例apple，可以如下使用，并且通过get()方法来获取apple引用。也可以再创建一个继承WeakReference的类来对Apple进行弱引用，下面就会使用这种方式。

```
WeakReference<Apple> appleWeakReference = new WeakReference<>(apple);
Apple apple2 = appleWeakReference.get();
```

- WeakReference(T referent, ReferenceQueue<? super T> q)：与上面的构造方法比较，多了个ReferenceQueue，在对象被回收后，会把弱引用对象，也就是WeakReference对象或者其子类的对象，放入队列ReferenceQueue中，注意不是被弱引用的对象，被弱引用的对象已经被回收了。

三. 使用WeakReference

下面是使用继承WeakReference的方式来使用软引用，并且不使用ReferenceQueue。

简单类Apple

```
package io.github.brightloong.lab.reference;

/**
 * Apple class
 *
 * @author BrightLoong
 * @date 2018/5/25
 */
public class Apple {

    private String name;

    public Apple(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    /**
     * 覆盖finalize，在回收的时候会执行。
     * @throws Throwable
     */
    @Override
    protected void finalize() throws Throwable {
        super.finalize();
        System.out.println("Apple： " + name + " finalize。");
    }

    @Override
    public String toString() {
        return "Apple{" +
                "name='" + name + '\'' +
                '}' + ", hashCode:" + this.hashCode();
    }
}
```

继承WeakReference的Salad

```
package io.github.brightloong.lab.reference;

import java.lang.ref.WeakReference;

/**
 * Salad class
 * 继承WeakReference，将Apple作为弱引用。
 * 注意到时候回收的是Apple，而不是Salad
 *
 * @author BrightLoong
 * @date 2018/5/25
 */
public class Salad extends WeakReference<Apple> {
    public Salad(Apple apple) {
        super(apple);
    }
}
```

Clent调用和输出

```
package io.github.brightloong.lab.reference;

import java.lang.ref.WeakReference;

/**
 * Main class
 *
 * @author BrightLoong
 * @date 2018/5/24
 */
public class Client {
    public static void main(String[] args) {
        Salad salad = new Salad(new Apple("红富士"));
        //通过WeakReference的get()方法获取Apple
        System.out.println("Apple:" + salad.get());
        System.gc();
        try {
            //休眠一下，在运行的时候加上虚拟机参数-XX:+PrintGCDetails，输出gc信息，确定gc发生了。
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        //如果为空，代表被回收了
        if (salad.get() == null) {
            System.out.println("clear Apple。");
        }
    }
}
```

输出如下：

```
Apple:Apple{name='红富士'}, hashCode:1846274136
[GC (System.gc()) [PSYoungGen: 3328K->496K(38400K)] 3328K->504K(125952K), 0.0035102 secs]
[Full GC (System.gc()) [PSYoungGen: 496K->0K(38400K)] [ParOldGen: 8K->359K(87552K)] 504K-
Apple: 红富士 finalize。
clear Apple。
```

ReferenceQueue的使用

```
package io.github.brightloong.lab.reference;

import java.lang.ref.Reference;
import java.lang.ref.ReferenceQueue;
import java.lang.ref.WeakReference;

/**
 * Client2 class
 *
 * @author BrightLoong
 * @date 2018/5/27
 */
public class Client2 {
    public static void main(String[] args) {
        ReferenceQueue<Apple> appleReferenceQueue = new ReferenceQueue<>();
        WeakReference<Apple> appleWeakReference = new WeakReference<Apple>(new Apple("青
        WeakReference<Apple> appleWeakReference2 = new WeakReference<Apple>(new Apple("毒

        System.out.println("=====gc调用前=====");
        Reference<? extends Apple> reference = null;
        while ((reference = appleReferenceQueue.poll()) != null ) {
            //不会输出，因为没有回收被弱引用的对象，并不会加入队列中
            System.out.println(reference);
        }
        System.out.println(appleWeakReference);
        System.out.println(appleWeakReference2);
        System.out.println(appleWeakReference.get());
        System.out.println(appleWeakReference2.get());

        System.out.println("=====调用gc=====");
        System.gc();
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("=====gc调用后=====");

        //下面两个输出为null,表示对象被回收了
        System.out.println(appleWeakReference.get());
        System.out.println(appleWeakReference2.get());

        //输出结果，并且就是上面的appleWeakReference、appleWeakReference2，再次证明对象被回收了
        Reference<? extends Apple> reference2 = null;
        while ((reference2 = appleReferenceQueue.poll()) != null ) {
            //如果使用继承的方式就可以包含其他信息了
            System.out.println("appleReferenceQueue中：" + reference2);
        }
    }
}
```

结果输出如下：

```
=====gc调用前=====
java.lang.ref.WeakReference@6e0be858
java.lang.ref.WeakReference@61bbe9ba
Apple{name='青苹果'}, hashCode:1627674070
Apple{name='毒苹果'}, hashCode:1360875712
=====调用gc=====
Apple:  毒苹果 finalize。
Apple:  青苹果 finalize。
=====gc调用后=====
null
null
appleReferenceQueue中:java.lang.ref.WeakReference@6e0be858
appleReferenceQueue中:java.lang.ref.WeakReference@61bbe9ba

Process finished with exit code 0
```

可以看到在队列中（ReferenceQueue），调用gc之前是没有内容的，调用gc之后，对象被回收了，并且弱引用对象appleWeakReference和appleWeakReference2被放入了队列中。

关于其他三种引用，强引用、软引用、虚引用，可以参考 http://www.cnblogs.com/gudi/p/6403953.html (http://www.cnblogs.com/gudi/p/6403953.html)