Learning the code way

A small step towards sharing what I have learned.

Search

Monday, 24 February 2014

Reference Queues

In the previous posts we had a look at weak and soft references. The java.lang.ref package also includes a class **ReferenceQueue**. As per the class documentation:

```
Reference queues, to which registered reference objects are appended by the garbage
collector after the appropriate reachability changes are detected.
```

What does a reachability change mean ?
We know that the Garbage collector will free up any weak reference objects or weakly reachable objects as a part of the garbage collection process. The flow for the same is:
If the garbage collector discovers an object that is weakly reachable, the following occurs:

1. The WeakReference object's referent field is set to null, thereby making it not refer to the heap object any longer.
2. The heap object that had been referenced by the WeakReference is declared finalizable.
3. The WeakReference object is added to its ReferenceQueue. Then the heap object's finalize() method is run and its memory freed.

So what is this **ReferenceQueue** ?
When creating non-strong reference objects we have the option of passing a reference queue as a part of the Reference constructor. As seen from the above explanation, **this reference queue is a way for the GC to inform the program that a certain object is no longer reachable**. Consider the below example:

```
public static void main(String[] args) throws InterruptedException {
      SavePoint savePoint = new SavePoint("Random"); // a strong object

      ReferenceQueue<SavePoint> savepointQ = new ReferenceQueue<SavePoint>();// the ReferenceQueue
      WeakReference<SavePoint> savePointWRefernce = new WeakReference<SavePoint>(savePoint, savepointQ);

      System.out.println("SavePoint created as a weak ref " + savePointWRefernce);
      Runtime.getRuntime().gc();
      System.out.println("Any weak references in Q ? " + (savepointQ.poll() != null));
      savePoint = null; // the only strong reference has been removed. The heap
                        // object is now only weakly reachable

      System.out.println("Now to call gc...");
      Runtime.getRuntime().gc(); // the object will be cleared here - finalize will be called.

      System.out.println("Any weak references in Q ? " + (savepointQ.remove() != null));
      System.out.println("Does the weak reference still hold the heap object ? " + (savePointWRefernce.get() != null));
      System.out.println("Is the weak reference added to the ReferenceQ  ? " + (savePointWRefernce.isEnqueued()));

   }
```

The program :

1. Creates a strong reference and adds it to a Weak reference savePointWRefernce. The object in memory is now referenced by a strong reference and a weak reference - hence strongly reachable.
2. The first call to garbage collector will not clear our savepoint object as it is a strong reference. Hence the poll method of the referenceQ will return null. (poll method is non - blocking it checks and returns immediately.)
3. The savePoint reference variable is set to null. Our heap object is now referenced only by the weak reference - hence it is weakly reachable.
4. The second gc call will now locate the object, executes its finalize method and mark this object to be freed. The object is also added to the ReferenceQ.
5. A call to the remove method of the ReferenceQ will return the object. remove is a blocking method. it will wait till an object has been made available in the Queue. (poll method might not work as the recycling process is happening on a separate thread.)

The output is as below:

```
SavePoint created as a weak ref java.lang.ref.WeakReference@19821f
Any weak references in Q ? false
Now to call gc...
Any weak references in Q ? true
Does the weak reference still hold the heap object ? false
Is the weak reference added to the ReferenceQ  ? false
```

As seen once the object was ready to be released it was added to the reference queue. So the reference queue is like a callback to our java program, telling us that a particular object is released from its reference and is not available to our code anymore.

Can we re-attach an object in the reference Q ?
We know that the finalize method can be used to re-attach objects. make them strongly reachable - e.g. by adding them to a static list. The object will now not be reclaimed in this cycle. Let us try to reclaim the object:

```
public static void main(String[] args) throws InterruptedException {
      SavePoint savePoint = new SavePoint("Random"); // a strong object

      ReferenceQueue<SavePoint> savepointQ = new ReferenceQueue<SavePoint>();// the ReferenceQueue
      WeakReference<SavePoint> savePointWRefernce = new WeakReference<SavePoint>(savePoint, savepointQ);

      System.out.println("Any weak references in Q ? " + (savepointQ.poll() != null));
      savePoint = null;

      System.out.println("Now to call gc...");
      Runtime.getRuntime().gc(); // the object will be cleared here - finalize Swill be called.
      Reference<? extends SavePoint> reCreatedSavePoint = savepointQ.remove();
      System.out.println("Any weak references in Q ? " + (reCreatedSavePoint != null));
      System.out.println("Is this same as original weak reference ? " + (reCreatedSavePoint == savePointWRefernce));
      System.out.println(" and heap object is " + reCreatedSavePoint.get());
   }
```

The above program tries to retrieve the freed object from the reference queue. The output is as below:

```
Any weak references in Q ? false
Now to call gc...
Any weak references in Q ? true
Is this same as original weak reference ? true
 and heap object is null
```

As seen from above the Reference queue actually holds within it the WeakReference which lost its heap object to clean up. The WeakReference does not have any association to the memory object. The get call above returns null. **Unlike with finalize when we can make the object alive again, with the ReferenceQ there is no way to reach the released java object**. Reference Queues are just for References that got freed by garbage collection. They cannot be used to make alive our objects again. They can only be used to notify our code about the loss of memory objects referred to by these non- strong references.

Posted by Robin Varghese at 22:50

Labels: java api

No comments:

Post a comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)