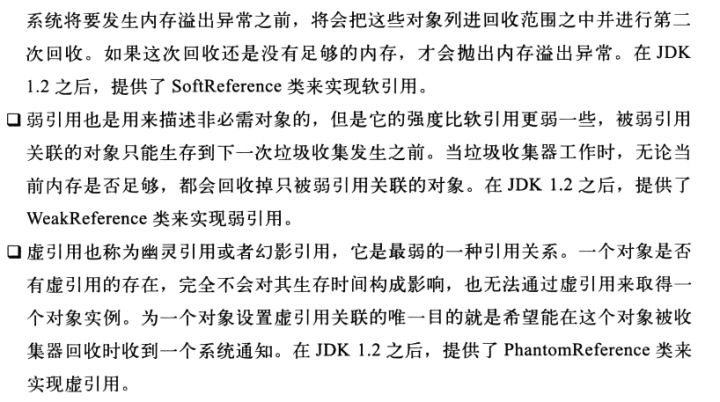
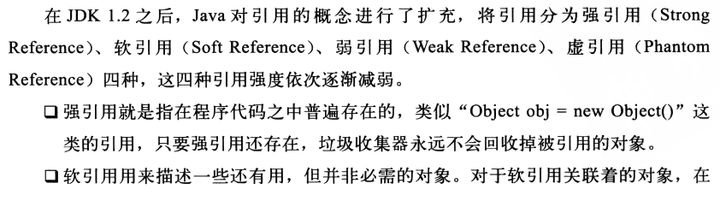
[**wuxinliulei**](https://www.zhihu.com/people/guo-wu-xin)

做好自己

24 人赞同了该回答



-----------------------

理论再多不如应用来的实际；

WeakReference使用实例：

ThreadLocalMap类 该类时ThreadLocal的静态内部类，该Map使用开放地址法处理hash冲突的Map类，key为ThreadLocal对象，value为TheadLocal对象所对应的值value；

其中Entry对象当中的Key值对TheadLocal的引用就是WeakReference

**static** **class** **Entry** **extends** WeakReference**<**ThreadLocal**>**

**{**

*/\* The value assocaiated with this ThreadLocal.\*/*

Object value**;**

Entry**(**ThreadLocal k**,**Object v**)**

**{**

**super(**k**);**

value **=** v**;**

**}**

**}**

这样当ThreadLocal对象除了Entry对象外没有其他引用的时候，在下一次垃圾回收发生时，该对象将被回收；

这也就导致在使用Entry对象获取key值得时候，需要判断是否为空，如果为空，则说明已经被回收了，此时将value值手动清除即可；

其实因为ThreadLocal牵涉到线程本地变量的操作，对于对象何时被清除，程序逻辑一般不太好实现，所以JDK设计者将其设置为了自动清除，

其实大部分TheadLocal对象我们都将其设置为private static 对象，一般不会被弱引用清除掉；

具体更为详细的内容请查询ThreadLocal

------------------------

SoftReference

**public** Method **getMethod(**String name**,** Class**<?>...** parameterTypes**)**

**throws** NoSuchMethodException**,** SecurityException **{**

*// be very careful not to change the stack depth of this*

*// checkMemberAccess call for security reasons*

*// see java.lang.SecurityManager.checkMemberAccess*

checkMemberAccess**(**Member**.**PUBLIC**,** Reflection**.**getCallerClass**(),** **true);**

Method method **=** getMethod0**(**name**,** parameterTypes**);**

**if** **(**method **==** **null)** **{**

**throw** **new** NoSuchMethodException**(**getName**()** **+** "." **+**

name **+** argumentTypesToString**(**parameterTypes**));**

**}**

**return** method**;**

**}**

下面看调用的getMethod0

**private** Method **getMethod0(**String name**,** Class**<?>[]** parameterTypes**)** **{**

*// Note: the intent is that the search algorithm this routine*

*// uses be equivalent to the ordering imposed by*

*// privateGetPublicMethods(). It fetches only the declared*

*// public methods for each class, however, to reduce the*

*// number of Method objects which have to be created for the*

*// common case where the method being requested is declared in*

*// the class which is being queried.*

Method res**;**

*// Search declared public methods*

**if** **((**res **=** searchMethods**(**privateGetDeclaredMethods**(true),**

name**,**

parameterTypes**))** **!=** **null)** **{**

**return** res**;**

**}**

*// Search superclass's methods*

**if** **(!**isInterface**())** **{**

Class**<?** **super** T**>** c **=** getSuperclass**();**

**if** **(**c **!=** **null)** **{**

**if** **((**res **=** c**.**getMethod0**(**name**,** parameterTypes**))** **!=** **null)** **{**

**return** res**;**

**}**

**}**

**}**

*// Search superinterfaces' methods*

Class**<?>[]** interfaces **=** getInterfaces**();**

**for** **(int** i **=** 0**;** i **<** interfaces**.**length**;** i**++)** **{**

Class**<?>** c **=** interfaces**[**i**];**

**if** **((**res **=** c**.**getMethod0**(**name**,** parameterTypes**))** **!=** **null)** **{**

**return** res**;**

**}**

**}**

*// Not found*

**return** **null;**

**}**

下面看调用的privateGetDeclaredMethods

*/\*\**

*\* 获取类中声明方法的具体实现*

*\* @param publicOnly 是否只获取public方法*

*\* @return 方法数组*

*\*/*

**private** Method**[]** **privateGetDeclaredMethods(boolean** publicOnly**)** **{**

*// 等待系统内部类初始化完成，系统属性(sun.reflect.noCaches)被解析完成*

checkInitted**();**

Method**[]** res **=** **null;**

*// 根据用户指定的系统属性sun.reflect.noCaches值来决定是否使用缓存，默认使用缓存*

**if** **(**useCaches**)** **{**

*// 清空缓存，将缓存的declaredFields、declaredMethods、annotations等设置为null*

clearCachesOnClassRedefinition**();**

*// 如果只获取public方法*

**if** **(**publicOnly**)** **{**

*// 如果declaredPublicFields缓存可用，则直接从缓存中获取*

**if** **(**declaredPublicMethods **!=** **null)** **{**

res **=** **(**Method**[])** declaredPublicMethods**.**get**();**

**}**

**}** **else** **{**

*// 如果declaredMethods缓存可用，则直接从缓存中获取*

**if** **(**declaredMethods **!=** **null)** **{**

res **=** **(**Method**[])** declaredMethods**.**get**();**

**}**

**}**

**if** **(**res **!=** **null)** **return** res**;**

**}**

*// 不使用缓存，则需要调用本地方法进行获取*

res **=** getDeclaredMethods0**(**publicOnly**);**

*// 如果可以使用缓存，则设置缓存，以备下次使用*

**if** **(**useCaches**)** **{**

**if** **(**publicOnly**)** **{**

declaredPublicMethods **=** **new** SoftReference**(**res**);**

**}** **else** **{**

declaredMethods **=** **new** SoftReference**(**res**);**

**}**

**}**

**return** res**;**

**}**

其中 declaredPublicMethods就是一个SoftReference

我们来看下这个成员变量的定义

private volatile transient SoftReference declaredPublicMethods;

private volatile transient SoftReference declaredMethods;

**if(**publicOnly**){**

*// 如果declaredPublicFields缓存可用，则直接从缓存中获取*

**if(**declaredPublicMethods **!=** **null){**

res **=(**Method**[])** declaredPublicMethods**.**get**();**

**}**

**}else{**

*// 如果declaredMethods缓存可用，则直接从缓存中获取*

**if(**declaredMethods **!=** **null){**

res **=(**Method**[])** declaredMethods**.**get**();**

**}**

**}**

其中res就是要获取的Method[] res 方法数组

如果发现get的结果已经为空，说明没有初始化或者已经被GC掉了，就重新获取一份新的并设置即可；

所以SoftReference非常适合用于做不是很重要的数据缓存，当内存紧张时，可以被GC掉

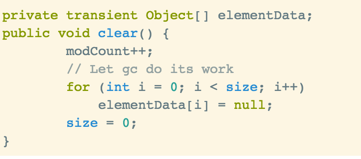
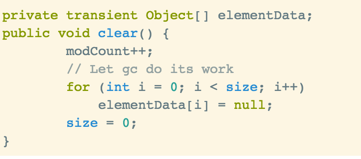
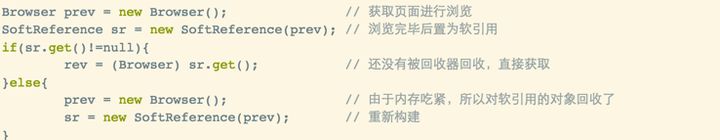
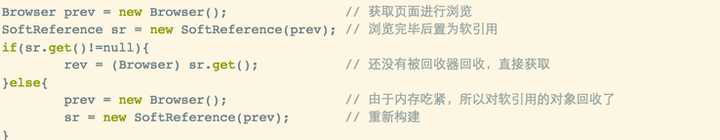
其实SoftReference和WeakReference都经常用来作为缓存来使用，不过WeakReference更容易被清除而已。

下面的文章也有详细的介绍

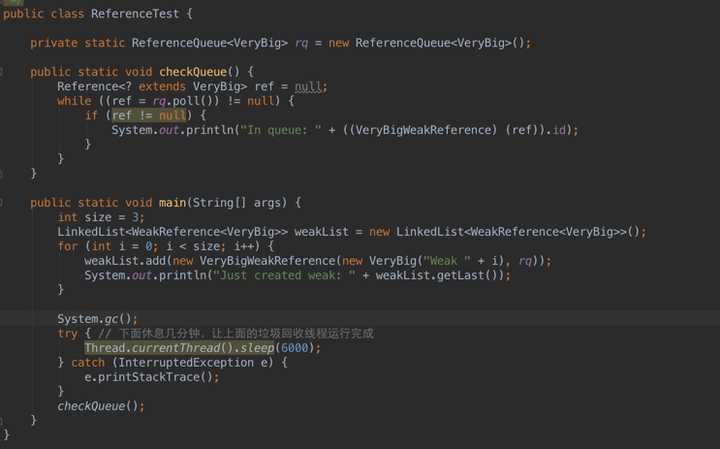
<https://www.zhihu.com/question/37401125>

作者：winterSunshine  
链接：https://www.zhihu.com/question/37401125/answer/100981172  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

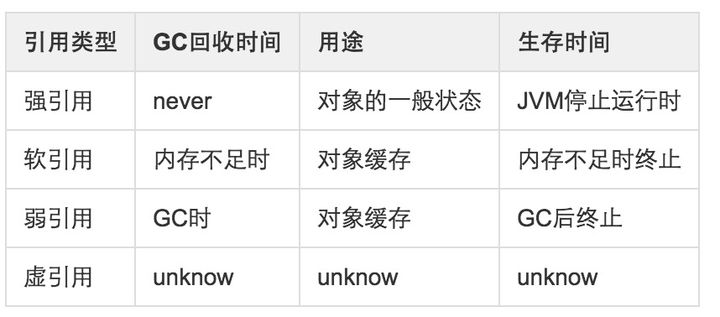
1.概念解释

* **强引用**是使用最普遍的引用：Object o=new Object(); 特点：不会被GC
  + 将对象的引用显示地置为null：o=null; // 帮助垃圾收集器回收此对象
  + 举例ArrayList的实现源代码：
* **软引用**用来描述一些还有用但是并非必须的对象，在Java中用java.lang.ref.SoftReference类来表示。对于软引用关联着的对象，只有在内存不足的时候JVM才会回收该对象。因此，这一点可以很好地用来解决OOM的问题，并且这个特性很适合用来实现缓存：比如网页缓存、图片缓存等。
  + 浏览器网页缓存实例：  
    
  + 软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收器回收，Java虚拟机就会把这个软引用加入到与之关联的引用队列中。
* **弱引用**与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。

如果这个对象是偶尔的使用，并且希望在使用时随时就能获取到，但又不想影响此对象的垃圾收集，那么你应该用 Weak Reference 来记住此对象。

1. 实例：

* **虚引用**也称为幻影引用：一个对象是都有虚引用的存在都不会对生存时间都构成影响，也无法通过虚引用来获取对一个对象的真实引用。唯一的用处：能在对象被GC时收到系统通知，JAVA中用PhantomReference来实现虚引用。

2.对比不同:  


[编辑于 2016-05-15](//www.zhihu.com/question/37401125/answer/100981172)

​赞同 51​​4 条评论

​分享

​收藏​喜欢

​

收起​

继续浏览内容



知乎

发现更大的世界

打开



Chrome

继续

[](//www.zhihu.com/people/ted-mosby-4-95)

[Ted Mosby](//www.zhihu.com/people/ted-mosby-4-95)

9 人赞同了该回答

会有坑，不推荐使用，除非你真正理解并了解其实现。  
能做到这点的人全世界不超过100人。  
直到2015年底开发组还在讨论修改三者的定义描述。

[编辑于 2016-02-13](//www.zhihu.com/question/37401125/answer/85869021)

​赞同 9​​8 条评论

​分享

​收藏​喜欢

​

继续浏览内容



知乎

发现更大的世界

打开



Chrome

继续

[](//www.zhihu.com/people/cniao5)

[[已重置]](//www.zhihu.com/people/cniao5)

19 人赞同了该回答

先来一则关于「攻城狮们对弱引用认知度」相关的调查，如下是 「技术小黑屋」（[http://droidyue.com/](https://link.zhihu.com/?target=http%3A//droidyue.com/)）对于「Java高级开发工程师的应聘者对弱引用的认知度」的一些看法，看到此，我很是吃惊。

「不久之前，我面试了一些求职 Java 高级开发工程师的应聘者。我常常会面试他们说，“你能给我介绍一些 Java 中的弱引用吗？”，如果面试者说，“嗯，是不是垃圾回收有关的？”，我就会基本满意了，我并不期待回答是一篇诘究本末的论文描述。

然而事与愿违，我很吃惊的发现，在将近 20 多个有着平均 5 年开发经验和高学历背景的应聘者中，居然只有两个人知道弱引用的存在；但是在这两个人之中只有一个人真正了解这方面的知识。在面试过程中，我还尝试提示一些东西，来看看有没有人突然说一声“原来是这个啊”，结果很是让我失望。

我开始困惑，为什么这块的知识如此不被重视。毕竟**弱引用是一个很有用途的特性，况且这个特性已经在 7 年前 Java 1.2 发布时便引入了**。

很庆幸，在我还没从事工作之前就知道了弱引用，并有幸学习它，心里窃喜。

**一、Java 引用介绍**

从 java 1.2 版本引入 java.lang.ref 包，共 4 种引用，这 4 种引用的级别高低依次为：

强引用 > 软引用 > 弱引用 > 虚引用

**1.1、强引用 Strong Reference**

强引用是使用最普遍的引用。**如果一个对象具有强引用，那垃圾回收器绝不会回收它。**当内存空间不足，Java 虚拟机宁愿抛出 OutOfMemoryError 错误，使程序异常终止，也不会靠随意回收具有强引用的对象来解决内存不足的问题。

强引用就是我们常使用到的引用，写法如下：

A a = new A();

B b = new B(a);

上面两个强引用就这样产生了，并且 a 是对象 A 的引用，b 是对象 B 的引用，而 B 还依赖于 A，那么就认为 B 是可以到达 A 的。

A a = new A();

B b = new B(a);

a = null;

当把 a = null 时，这时 a 不再指向 A 的地址。按道理：当某个对象不再被其它对象引用的时候，会被 GC 回收，而 a = null 时，A 对象不能被回收，B 还依赖于 A，造成了内存泄漏。

强引用最重要的就是它能够让引用变强，这就决定了它和GC的交互，如果一个对象通过强引用链接可到达，它就不会被 GC 回收。当然，如果你不想让正在使用的对象被回收，这正是强引用的强大之处。

**看到上述示例强引用带来的内存泄漏问题，我们要如何避免？**

**1.2、弱引用 Weak Reference**

弱引用就是将对象留在内存的能力弱于强引用，使用 WeakReference

（[https://developer.android.com/reference/java/lang/ref/WeakReference.html](https://link.zhihu.com/?target=https%3A//developer.android.com/reference/java/lang/ref/WeakReference.html)）。垃圾回收器会帮你来决定引用的对象何时回收并且将对象从内存移除。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。

弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联的引用队列中。

A a = new A();

WeakReference b = new WeakReference(a);//B b = new B(a);a = null;

当 a = null 时，这个时候 A 只被弱引用依赖，GC 回立刻回收 A 对象，这就是弱引用的好处，避免内存泄漏。

**1.3、引用队列 ReferenceQueue**

引用队列 ReferenceQueue

（[https://developer.android.com/reference/java/lang/ref/ReferenceQueue.html](https://link.zhihu.com/?target=https%3A//developer.android.com/reference/java/lang/ref/ReferenceQueue.html)）配合 Reference 子类等使用，当引用对象所指向的内存空间被 GC 回收后，该引用对象则被追加到引用队列的末尾。根据下面的代码，说明只供 Reference 实例调用，且只能调用一次。

/\*\*

\* Enqueue the reference object on the receiver.

\* @param reference reference object to be enqueued.

\* @return true if the reference was enqueued.

\*/boolean enqueue(Reference<? extends T> reference) { synchronized (lock) { if (enqueueLocked(reference)) {

lock.notifyAll(); return true;

} return false;

}

}

**引用队列有如下实例方法**

当队列中出队一个元素时，若队列为空返回 null，否则返回队列。

/\*\*

\* Polls this queue to see if a reference object is available. If one is

\* available without further delay then it is removed from the queue and

\* returned. Otherwise this method immediately returns <tt>null</tt>.

\* @return A reference object, if one was immediately available,

\* otherwise <code>null</code>

\*/public Reference<? extends T> poll() {

synchronized (lock) { if (head == null) return null; return reallyPollLocked();

}

}//入队private Reference<? extends T> reallyPollLocked() { if (head != null) {

Reference<? extends T> r = head; if (head == tail) {

tail = null;

head = null;

} else {

head = head.queueNext;

} // Update queueNext to indicate that the reference has been

// enqueued, but is now removed from the queue.

r.queueNext = sQueueNextUnenqueued; return r;

} return null;

}

从队列中出队一个元素，若没有则阻塞直到有元素可出队或超过 timeout 指定的毫秒数（由于采用 wait(long timeout) 方式实现等待，因此时间不能保证）。

public Reference<? extends T> remove(long timeout)

throws IllegalArgumentException, InterruptedException

{ if (timeout < 0) { throw new IllegalArgumentException("Negative timeout value");

}

synchronized (lock) {

Reference<? extends T> r = reallyPollLocked(); if (r != null) return r; long start = (timeout == 0) ? 0 : System.nanoTime(); for (;;) { lock.wait(timeout);

r = reallyPollLocked(); if (r != null) return r; if (timeout != 0) { long end = System.nanoTime();

timeout -= (end - start) / 1000\_000; if (timeout <= 0) return null;

start = end;

}

}

}

}

**1.4、软引用 SoftReference**

软引用阻止 GC 回收其指向对象的能力要强一些。通过 SoftReference

（[https://developer.android.com/reference/java/lang/ref/WeakReference.html?q=SoftReference](https://link.zhihu.com/?target=https%3A//developer.android.com/reference/java/lang/ref/WeakReference.html%3Fq%3DSoftReference)）表示， 如果一个对象只具有软引用，则内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就会回收这些对象的内存。

**只要垃圾回收器没有回收它，该对象就可以被程序使用。**

当 JVM 中的内存不足的时候，垃圾回收器会释放那些只被软引用所指向的对象。如果全部释放完这些对象之后，内存还不足，才会抛出 OutOfMemoryError。

**软引用非常适合于创建缓存。**当系统内存不足的时候，缓存中的内容是可以被释放的。比如考虑一个图像编辑器的程序。该程序会把图像文件的全部内容都读取到内存中，以方便进行处理。而用户也可以同时打开多个文件。当同时打开的文件过多的时候，就可能造成内存不足。如果使用软引用来指向图像文件内容的话，垃圾回收器就可以在必要的时候回收掉这些内存。

软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收器回收，JVM 就会把这个软引用加入到与之关联的引用队列中。

**软引用与弱引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。**在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。

**1.5、虚引用 Phantom Reference**

“虚引用”顾名思义，就是形同虚设，与其他几种引用都不同，虚引用并不会决定对象的生命周期。

我们可以通过 get 方法来得到其指定对象，它的唯一作用就是当其指向的对象被回收之后，自己被加入到引用队列，用作记录该引用指向的对象已被销毁。**虚引用职能在其指向的对象从内存中移除掉之后才会加入到引用队列。**其 get 方法一直返回 null 就是为了阻止其指向的几乎被销毁的对象重新复活。

**虚引用主要用来跟踪对象被垃圾回收器回收的活动。**虚引用与软引用和弱引用的一个区别在于：虚引用必须和引用队列 （ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之 关联的引用队列中。

ReferenceQueue queue = new ReferenceQueue ();

PhantomReference pr = new PhantomReference (object, queue);

程序可以通过判断引用队列中是否已经加入了虚引用，来了解被引用的对象是否将要被垃圾回收。如果程序发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。

**虚引用使用场景主要有两个。**

**1、它允许你知道具体何时其引用的对象从内存中移除。**而实际上这是Java中唯一的方式。这一点尤其表现在处理类似图片的大文件的情况。当你确定一个图片数据对象应该被回收，你可以利用虚引用来判断这个对象回收之后在继续加载下一张图片。这样可以尽可能地避免可怕的内存溢出错误。

**2、虚引用可以避免很多析构时的问题。**finalize 方法可以通过创建强引用指向快被销毁的对象来让这些对象重新复活。然而，一个重写了 finalize 方法的对象如果想要被回收掉，需要经历两个单独的垃圾收集周期。在第一个周期中，某个对象被标记为可回收，进而才能进行析构。

但是因为在析构过程中仍有微弱的可能这个对象会重新复活。这种情况下，在这个对象真实销毁之前，垃圾回收器需要再次运行。因为析构可能并不是很及时，所以在调用对象的析构之前，需要经历数量不确定的垃圾收集周期。这就意味着在真正清理掉这个对象的时候可能发生很大的延迟。这就是为什么当大部分堆被标记成垃圾时还是会出现烦人的内存溢出错误。

**引用总结**



**二、CountDownTimer 中使用到弱引用讲**

上文中，在倒计时 Demo 中，用到了弱引用。看完弱引用之后，你应该对弱引用有了大致了解。

那么下面针对上一章内容，我们再回过头来看看**为什么要用弱引用吧**。我们是要用 TextView 来实现倒计时功能，如果直接用强引用，TextView 在被其它对象引用不能及时回收时，可能会造成内存泄漏。

/\*\*

\* 在按钮上启动一个定时器

\* @param tvVerifyCode 验证码控件

\* @param defaultString 按钮上默认的字符串

\* @param max 失效时间（单位：s）

\* @param interval 更新间隔（单位：s）

\*/public static void startTimer(final WeakReference<TextView> tvVerifyCode, final String defaultString, int max, int interval) {

tvVerifyCode.get().setEnabled(false); new CountDownTimer(max \* 1000, interval \* 1000 - 10) { @Override

public void onTick(long time) { if (null == tvVerifyCode.get()) this.cancel(); else

tvVerifyCode.get().setText("" + ((time + 15) / 1000) + "s");

} @Override

public void onFinish() { if (null == tvVerifyCode.get()) { this.cancel(); return;

}

tvVerifyCode.get().setEnabled(true);

tvVerifyCode.get().setText(defaultString);

}

}.start();

}

**参考：** [http://www.cnblogs.com/fsjohnhuang/p/4268411.html](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/fsjohnhuang/p/4268411.html) [http://blog.csdn.net/matrix\_xu/article/details/8424038](https://link.zhihu.com/?target=http%3A//blog.csdn.net/matrix_xu/article/details/8424038) [http://www.cnblogs.com/skywang12345/p/3154474.html](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/skywang12345/p/3154474.html) [http://droidyue.com/blog/2014/10/12/understanding-weakreference-in-java/](https://link.zhihu.com/?target=http%3A//droidyue.com/blog/2014/10/12/understanding-weakreference-in-java/) [http://www.infoq.com/cn/articles/cf-java-garbage-references](https://link.zhihu.com/?target=http%3A//www.infoq.com/cn/articles/cf-java-garbage-references)

本文为菜鸟窝作者蒋志碧的连载。“从 0 开始开发一款直播 APP ”系列来聊聊时下最火的直播 APP，如何完整的实现一个类”腾讯直播”的商业化项目。  
  
视频地址：[http://www.cniao5.com/course/10121](https://link.zhihu.com/?target=http%3A//www.cniao5.com/course/10121)

**（菜鸟窝-程序猿的黄埔军校。 如需转载，请注明出处）**

**加入直播APP交流群方式：**

识别下方二维码☟，添加菜鸟窝运营微信为好友，并备注【直播】，即可拉你入群