第4讲 | 强引用、软引用、弱引用、幻象引用有什么区别?

2018-05-12 杨晓峰

第4讲 | 强引用、软引用、弱引用、幻象引用有什么区别?
朗读人：黄洲君　10′23″ | 4.76M

在 Java 语言中，除了原始数据类型的变量，其他所有都是所谓的引用类型，指向各种不同的对象，理解引用对于掌握 Java 对象生命周期和 JVM 内部相关机制非常有帮助。

今天我要问你的问题是，强引用、软引用、弱引用、幻象引用有什么区别？具体使用场景是什么？

典型回答

不同的引用类型，主要体现的是对象不同的可达性（reachable）状态和对垃圾收集的影响。

所谓强引用（"Strong" Reference），就是我们最常见的普通对象引用，只要还有强引用指向一个对象，就能表明对象还“活着”，垃圾收集器不会碰这种对象。对于一个普通的对象，如果没有其他的引用关系，只要超过了引用的作用域或者显式地将相应（强）引用赋值为 null，就是可以被垃圾收集的了，当然具体回收时机还是要看垃圾收集策略。

软引用（SoftReference），是一种相对强引用弱化一些的引用，可以让对象豁免一些垃圾收集，只有当 JVM 认为内存不足时，才会去试图回收软引用指向的对象。JVM 会确保在抛出 OutOfMemoryError 之前，清理软引用指向的对象。软引用通常用来实现内存敏感的缓存，如

果还有空闲内存，就可以暂时保留缓存，当内存不足时清理掉，这样就保证了使用缓存的同时，不会耗尽内存。

弱引用（WeakReference）并不能使对象豁免垃圾收集，仅仅是提供一种访问在弱引用状态下对象的途径。这就可以用来构建一种没有特定约束的关系，比如，维护一种非强制性的映射关系，如果试图获取时对象还在，就使用它，否则重现实例化。它同样是很多缓存实现的选择。

对于幻象引用，有时候也翻译成虚引用，你不能通过它访问对象。幻象引用仅仅是提供了一种确保对象被 finalize 以后，做某些事情的机制，比如，通常用来做所谓的 Post-Mortem 清理机制，我在专栏上一讲中介绍的 Java 平台自身 Cleaner 机制等，也有人利用幻象引用监控对象的创建和销毁。

考点分析

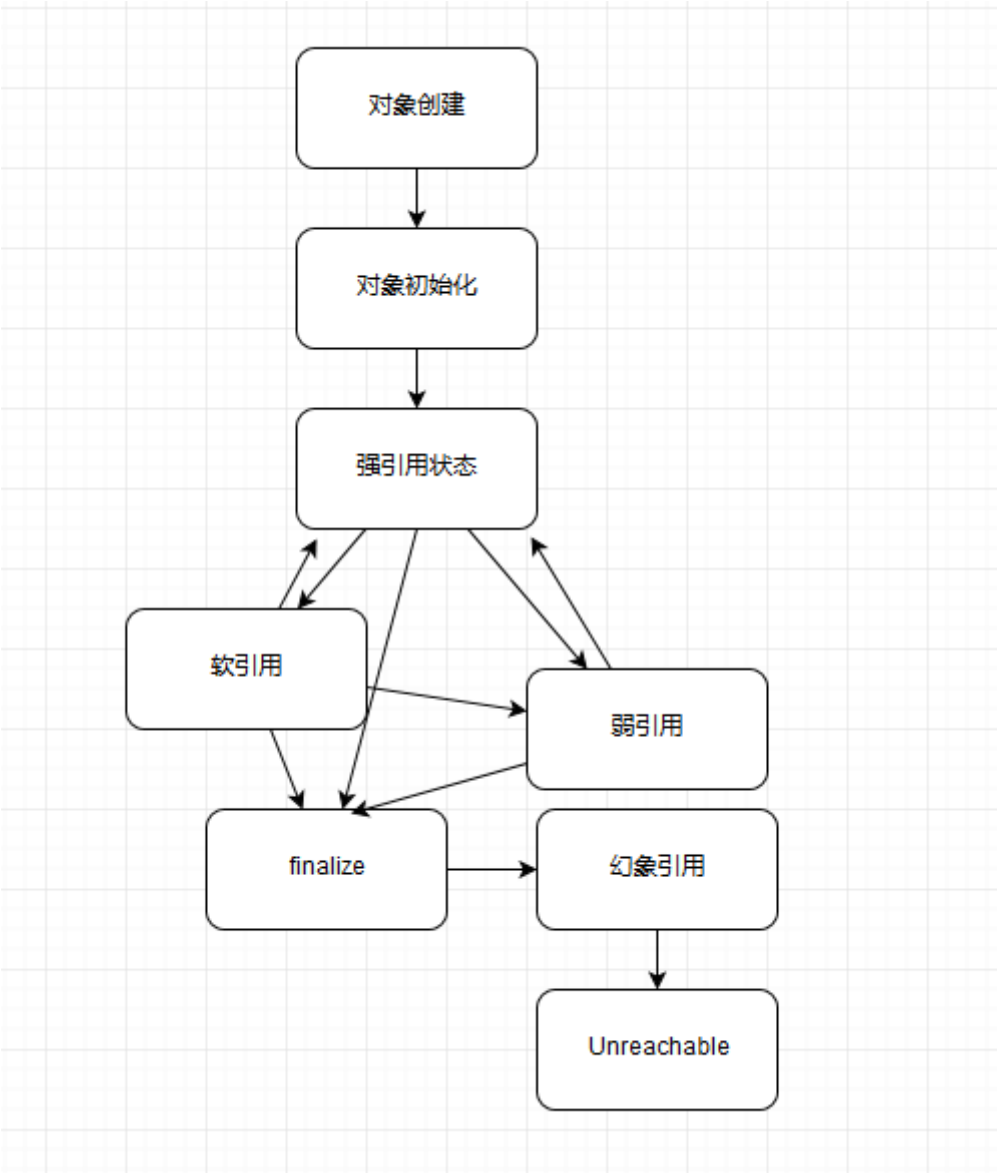
这道面试题，属于既偏门又非常高频的一道题目。说它偏门，是因为在大多数应用开发中，很少直接操作各种不同引用，虽然我们使用的类库、框架可能利用了其机制。它被频繁问到，是因为这是一个综合性的题目，既考察了我们对基础概念的理解，也考察了对底层对象生命周期、垃圾收集机制等的掌握。

充分理解这些引用，对于我们设计可靠的缓存等框架，或者诊断应用 OOM 等问题，会很有帮助。比如，诊断 MySQL connector-j 驱动在特定模式下（useCompression=true）的内存泄漏问题，就需要我们理解怎么排查幻象引用的堆积问题。

知识扩展

1. 对象可达性状态流转分析

首先，请你看下面流程图，我这里简单总结了对象生命周期和不同可达性状态，以及不同状态可能的改变关系，可能未必 100% 严谨，来阐述下可达性的变化。



我来解释一下上图的具体状态，这是 Java 定义的不同可达性级别（reachability level），具体如下：

- 强可达（Strongly Reachable），就是当一个对象可以有一个或多个线程可以不通过各种引用访问到的情况。比如，我们新创建一个对象，那么创建它的线程对它就是强可达。
- 软可达（Softly Reachable），就是当我们只能通过软引用才能访问到对象的状态。
- 弱可达（Weakly Reachable），类似前面提到的，就是无法通过强引用或者软引用访问，只能通过弱引用访问时的状态。这是十分临近 finalize 状态的时机，当弱引用被清除的时候，就符合 finalize 的条件了。
- 幻象可达（Phantom Reachable），上面流程图已经很直观了，就是没有强、软、弱引用关联，并且 finalize 过了，只有幻象引用指向这个对象的时候。
- 当然，还有一个最后的状态，就是不可达（unreachable），意味着对象可以被清除了。

判断对象可达性，是 JVM 垃圾收集器决定如何处理对象的一部分考虑。

所有引用类型，都是抽象类 java.lang.ref.Reference 的子类，你可能注意到它提供了 get() 方法：

T	get()	Returns this reference object's referent.

除了幻象引用（因为 get 永远返回 null），如果对象还没有被销毁，都可以通过 get 方法获取原有对象。这意味着，利用软引用和弱引用，我们可以将访问到的对象，重新指向强引用，也就是人为的改变了对象的可达性状态！这也是为什么我在上面图里有些地方画了双向箭头。

所以，对于软引用、弱引用之类，垃圾收集器可能会存在二次确认的问题，以保证处于弱引用状态的对象，没有改变为强引用。

但是，你觉得这里有没有可能出现什么问题呢？

不错，如果我们错误的保持了强引用（比如，赋值给了 static 变量），那么对象可能就没有机会变回类似弱引用的可达性状态了，就会产生内存泄漏。所以，检查弱引用指向对象是否被垃圾收集，也是诊断是否有特定内存泄漏的一个思路，如果我们的框架使用到弱引用又怀疑有内存泄漏，就可以从这个角度检查。

2. 引用队列（ReferenceQueue）使用

谈到各种引用的编程，就必然要提到引用队列。我们在创建各种引用并关联到响应对象时，可以选择是否需要关联引用队列，JVM 会在特定时机将引用 enqueue 到队列里，我们可以从队列里获取引用（remove 方法在这里实际是有获取的意思）进行相关后续逻辑。尤其是幻象引用，get 方法只返回 null，如果再不指定引用队列，基本就没有意义了。看看下面的示例代码。利用引用队列，我们可以在对象处于相应状态时（对于幻象引用，就是前面说的被 finalize 了，处于幻象可达状态），执行后期处理逻辑。

```
Object counter = new Object();
ReferenceQueue refQueue = new ReferenceQueue<>();
PhantomReference<Object> p = new PhantomReference<>(counter, refQueue);
counter = null;
System.gc();
try {
    // Remove 是一个阻塞方法，可以指定 timeout，或者选择一直阻塞
    Reference<Object> ref = refQueue.remove(1000L);
    if (ref != null) {
        // do something
    }
} catch (InterruptedException e) {
```

```
    // Handle it
}
```

3. 显式地影响软引用垃圾收集

前面泛泛提到了引用对垃圾收集的影响，尤其是软引用，到底 JVM 内部是怎么处理它的，其实并不是非常明确。那么我们能不能使用什么方法来影响软引用的垃圾收集呢?

答案是有的。软引用通常会在最后一次引用后，还能保持一段时间，默认值是根据堆剩余空间计算的（以 M bytes 为单位）。从 Java 1.3.1 开始，提供了 -XX:SoftRefLRUPolicyMSPerMB 参数，我们可以以毫秒（milliseconds）为单位设置。比如，下面这个示例就是设置为 3 秒（3000 毫秒）。

```
-XX:SoftRefLRUPolicyMSPerMB=3000
```

这个剩余空间，其实会受不同 JVM 模式影响，对于 Client 模式，比如通常的 Windows 32 bit JDK，剩余空间是计算当前堆里空闲的大小，所以更加倾向于回收；而对于 server 模式 JVM，则是根据 -Xmx 指定的最大值来计算。

本质上，这个行为还是个黑盒，取决于 JVM 实现，即使是上面提到的参数，在新版的 JDK 上也未必有效，另外 Client 模式的 JDK 已经逐步退出历史舞台。所以在我们应用时，可以参考类似设置，但不要过于依赖它。

4. 诊断 JVM 引用情况

如果你怀疑应用存在引用（或 finalize）导致的回收问题，可以有很多工具或者选项可供选择，比如 HotSpot JVM 自身便提供了明确的选项（PrintReferenceGC）去获取相关信息，我指定了下面选项去使用 JDK 8 运行一个样例应用：

```
-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintReferenceGC
```

这是 JDK 8 使用 ParrallelGC 收集的垃圾收集日志，各种引用数量非常清晰。

```
0.403: [GC (Allocation Failure) 0.871: [SoftReference, 0 refs, 0.0000393 secs]0.871: [WeakRef
```

注意：JDK 9 对 JVM 和垃圾收集日志进行了广泛的重构，类似 PrintGCTimeStamps 和 PrintReferenceGC 已经不再存在，我在专栏后面的垃圾收集主题里会更加系统的阐述。

5.Reachability Fence

除了我前面介绍的几种基本引用类型，我们也可以通过底层 API 来达到强引用的效果，这就是所谓的设置reachability fence。

为什么需要这种机制呢？考虑一下这样的场景，按照 Java 语言规范，如果一个对象没有指向强引用，就符合垃圾收集的标准，有些时候，对象本身并没有强引用，但是也许它的部分属性还在被使用，这样就导致诡异的问题，所以我们需要一个方法，在没有强引用情况下，通知 JVM 对象是在被使用的。说起来有点绕，我们来看看 Java 9 中提供的案例。

```
class Resource {
 private static ExternalResource[] externalResourceArray = ...
 int myIndex; Resource(...) {

     myIndex = ...
     externalResourceArray[myIndex] = ...;
     ...
 }
 protected void finalize() {
     externalResourceArray[myIndex] = null;
     ...
 }
 public void action() {
 try {
     // 需要被保护的代码
     int i = myIndex;
     Resource.update(externalResourceArray[i]);
 } finally {
     // 调用 reachbilityFence，明确保障对象 strongly reachable
     Reference.reachabilityFence(this);
 }
 }
 private static void update(ExternalResource ext) {
    ext.status = ...;
 }
}
```

方法 action 的执行，依赖于对象的部分属性，所以被特定保护了起来。否则，如果我们在代码中像下面这样调用，那么就可能会出现困扰，因为没有强引用指向我们创建出来的 Resource 对象，JVM 对它进行 finalize 操作是完全合法的。

```
new Resource().action()
```

类似的书写结构，在异步编程中似乎是很普遍的，因为异步编程中往往不会用传统的“执行 -> 返回 -> 使用”的结构。

在 Java 9 之前，实现类似类似功能相对比较繁琐，有的时候需要采取一些比较隐晦的小技巧。幸好，java.lang.ref.Reference 给我们提供了新方法，它是 JEP 193: Variable Handles 的一部分，将 Java 平台底层的一些能力暴露出来：

```
static void reachabilityFence(Object ref)
```

在 JDK 源码中，reachabilityFence 大多使用在 Executors 或者类似新的 HTTP/2 客户端代码中，大部分都是异步调用的情况。编程中，可以按照上面这个例子，将需要 reachability 保障的代码段利用 try-finally 包围起来，在 finally 里明确声明对象强可达。

今天，我总结了 Java 语言提供的几种引用类型、相应可达状态以及对于 JVM 工作的意义，并分析了引用队列使用的一些实际情况，最后介绍了在新的编程模式下，如何利用 API 去保障对象不被以为意外回收，希望对你有所帮助。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？给你留一道练习题，你能从自己的产品或者第三方类库中找到使用各种引用的案例吗？它们都试图解决什么问题？

请你在留言区写写你的答案，我会选出经过认真思考的留言，送给你一份学习鼓励金，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享出去，或许你能帮到他。

精选留言

Miaozhe 2

接着上个问题：
老师，问个问题:我自己定义一个类，重写finalize方法后，创建一个对象，被幻想引用，同时该幻想对象使用ReferenceQueue。
当我这个对象指向null，被GC回收后，ReferenceQueue中没有改对象，不知道是什么原因？如果我把类中的finalize方法移除，ReferenceQueue就能获取被释放的对象。

2018-05-17作者回复文章图里阐明了，幻象引用enque发生在finalize之后，你查查是不是卡在FinalReference queue里了，那是实现finalization的地方

杨老师，我去查看了，Final reference和Reference发现是Reference Handle线程在监控，但是Debug进出去，还是没有搞清楚原理。

不过，我又发现类中自定义得Finalize,如果是空的，正常。如果类中有任何代码，都不能进入Reference Queue，怀疑是对象没有被GC回收。

2018-05-18

作者回复

空的Finalize实现，不会起作用的；

Finalizer是懒家伙，试试system.runfinalization；

2018-05-18

公号-Java大后端 146

在Java语言中，除了基本数据类型外，其他的都是指向各类对象的对象引用；Java中根据其生命周期的长短，将引用分为4类。

1 强引用

特点：我们平常典型编码Object obj = new Object()中的obj就是强引用。通过关键字new创建的对象所关联的引用就是强引用。 当JVM内存空间不足，JVM宁愿抛出OutOfMemoryError运行时错误（OOM），使程序异常终止，也不会靠随意回收具有强引用的“存活”对象来解决内存不足的问题。对于一个普通的对象，如果没有其他的引用关系，只要超过了引用的作用域或者显式地将相应（强）引用赋值为 null，就是可以被垃圾收集的了，具体回收时机还是要看垃圾收集策略。

2 软引用

特点：软引用通过SoftReference类实现。 软引用的生命周期比强引用短一些。只有当 JVM 认为内存不足时，才会去试图回收软引用指向的对象：即JVM 会确保在抛出 OutOfMemoryError 之前，清理软引用指向的对象。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。后续，我们可以调用ReferenceQueue的poll()方法来检查是否有它所关心的对象被回收。如果队列为空，将返回一个null,否则该方法返回队列中前面的一个Reference对象。

应用场景：软引用通常用来实现内存敏感的缓存。如果还有空闲内存，就可以暂时保留缓存，当内存不足时清理掉，这样就保证了使用缓存的同时，不会耗尽内存。

3 弱引用

弱引用通过WeakReference类实现。 弱引用的生命周期比软引用短。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。由于垃圾回收器是一个优先级很低的线程，因此不一定会很快回收弱引用的对象。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。

应用场景：弱应用同样可用于内存敏感的缓存。

4 虚引用

特点：虚引用也叫幻象引用，通过PhantomReference类来实现。无法通过虚引用访问对象的任何属性或函数。幻象引用仅仅是提供了一种确保对象被 finalize 以后，做某些事情的机制。如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。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一些程序行动。

应用场景：可用来跟踪对象被垃圾回收器回收的活动，当一个虚引用关联的对象被垃圾收集器回收之前会收到一条系统通知。

2018-05-12

作者回复

高手

2018-05-14

海怪哥哥 88

我的理解，java的这种抽象很有意思。
强引用就像大老婆，关系很稳固。
软引用就像二老婆，随时有失宠的可能，但也有扶正的可能。
弱引用就像情人，关系不稳定，可能跟别人跑了。
幻像引用就是梦中情人，只在梦里出现过。

2018-05-13

作者回复

牛

2018-05-14

Jerry银银 70

1. 强引用：项目中到处都是。

2. 软引用：图片缓存框架中，“内存缓存”中的图片是以这种引用来保存，使得JVM在发生OOM之前，可以回收这部分缓存

3. 虚引用：在静态内部类中，经常会使用虚引用。例如，一个类发送网络请求，承担callback的静态内部类，则常以虚引用的方式来保存外部类(宿主类)的引用，当外部类需要被JVM回收时，不会因为网络请求没有及时回来，导致外部类不能被回收，引起内存泄漏

4. 幽灵引用：这种引用的get()方法返回总是null，所以，可以想象，在平常的项目开发肯定用的少。但是根据这种引用的特点，我想可以通过监控这类引用，来进行一些垃圾清理的动作。不过具体的场景，还是希望峰哥举几个稍微详细的实战性的例子？

2018-05-12

作者回复

非常不错，高手；
你可以参考jdk内部cleaner使用，一个方面就贴太多，有凑字数嫌疑了

2018-05-14

肖一林 45

这篇文章只描述了强引用，软引用，弱引用，幻想引用的特征。没有讲他们的概念，更没有讲怎么用...gc roots也没提到。希望能补充完来龙去脉，让没有太多基础的人也能看懂

2018-05-14

探索无止境 18

希望可以配合一些实际的例子来讲解各种引用会更好，不会仅停留在理论理解层面，实际例子更有助于理解！

2018-05-14

作者回复

谢谢反馈，我会平衡一下，主要是贴太多代码很容易字数就满了，也不利于录音频

2018-05-14

Jane 16

引用出现的根源是由于GC内存回收的基本原理—GC回收内存本质上是回首对象，而目前比较流行的回收算法是可达性分析算法，从GC Roots开始按照一定的逻辑判断一个对象是否可达，不可达的话就说明这个对象已死（除此之外另外一种常见的算法就是引用计数法，但是这种算法有个问题就是不能解决相互引用的问题）。基于此Java向用户提供了四种可用的引用：即我们本章讲解到的几种，同时还提供了一种不可被使用的引用—FinalReference，这个引用是和析构函数密切相关的）。强引用，开发者可以通过new的方式创建，其它的几种引用Java提供了相应的类：SoftReference、WeakReference、PhantomReference。如果你去查看源码你会发现，这个类实现的核心是Reference与ReferenceQueue（更通俗地说引用队列）两个类，而且这两个类也特别的简单。Reference类似一个链表结构，通过创建一个守护线程来执行对应引用的清除、Cleaner.clean（如果传入的对象是该类的话）、以及引用的入队操作（需要在创建引用的时候制定一个引用队列）；ReferenceQueue这是制定了引用队列的一些具体操作，简单的来说它也是一个链表结构，并提供了一些基本的链表操作）。而除了强引用外其它的都是继承于此，通过这样的类约束了引用的相关内容，便于和GC进行交互。这几种引用的区别如下：

1:强引用是只有当GC明确判断该引用无效的时候才会回收相应的引用对象，即使抛出OOME警告。

2:软引用是当GC检测到继续创建对象会导致OOME的时候会进行一次垃圾回收，这次回收会讲软引用回收以防抛出异常，根据这样的特点该引用常用来被当作缓存使用。

3:虚引用是哪些如果引用未被使用，就会在最近的一次GC的时候被回收。例如Java的TheardLocal与动态代理都是基于这样的一个引用实现的，一般针对那些比较敏感的数据。

4:幻想引用是针对那些已经执行完析构函数之后，仍然需要在执行一些其它操作的对象：比如资源对象的关闭就可以用到这个引用。

2018-05-16

feitian 14

我觉得录音和文字可以不一样，不要兼顾这两者，录音内容应该远多于文字，就像PPT一样，讲述的人表述的会远多于文字体现出来的东西。所以不用为了录音方便考虑文字内容多少，文字尽量能不靠录音也是完整的，录音的内容会更丰富，但有些不好描述的部分，比如代码要配合文字一起看。

2018-05-15

作者回复

好建议，回头和极客反馈下

2018-05-15

石头狮子 12

对各种引用的理解，可以理解为对象对 jvm 堆内存的占用时长。对于对象可达性垃圾回收算法，可达性可以认为回收内存的标志。

1，强引用，只要对象引用可达，对象使用的内存就一直被占用。

2，软引用，对象使用的内存一直占用，直到 jvm 认为有必要回收内存。

3，弱引用，对象使用的内存一直占用，直到下一次 gc。

求赞。😂 😂 😂

2018-05-12

kugool 7

感觉自己的基础还很差 除了强引用 其它几个都不是很明白

2018-05-12

爱吃面的蝎子王 6

希望作者照顾层次化的读者，讲名词概念要有具体解释，并能举例一二帮助理解，不然看完依旧似懂非懂一知半解。

2018-05-16

作者回复

谢谢反馈，回头把必要概念加个链接或解释

2018-05-18

哈 4

看完以后，真的很不明白。有一种告诉你了一下概念却又没有用具体实际进行比较说明，概念性的东西太抽象看完一点印象都没有...希望作者能用改进一下

2018-05-23

龙猫猫猫猫 4

热评第一讲得比这文章还好

2018-05-18

程序猿的小浣熊 4

我觉得可以在github上托管事例代码，说到关键代码的时候，直接链接过去就好。这样就能丰富内容了。

2018-05-15

鸠摩智 4

这个确实是日常开发所接触不到的知识点，所以看起来挺费力的

2018-05-12

作者回复

未必是直接写，类似基础架构处理mysql oom就会用到了

2018-05-12

kyq叶鑫 2

看到第四讲了，每一讲都看到留言有朋友说看完之后还是懵懵的希望作者多提供实际例子，因为大家都是理工思维，不喜虚的，喜欢直接上干货，建议作者可以从读者背景方面改善文章内容，软硬兼并。

2018-06-09

kursk.ye 2

于是我google到了这篇文章,http://www.kdgregory.com/index.php?page=java.refobj，花了几天（真的是几天，不是几小时）才基本读完，基本理解这几个reference的概念和作用，从这个角度来讲非常感谢作者，如果不是本文的介绍，我还以为GC还是按照reference counter的原理处理，原来思路早变了。话说回来，《Java Reference Objects》真值得大家好好琢磨，相信可以回答很多人的问题，比如strong reference , soft reference , weak reference怎么互转，如果一个obj 已经 = null,就obj = reference.get()呗，再有，文章中用weak reference 实现 canonicalizing map改善内存存储效率，减小存储空间的例子，真是非常经典啊。也希望作者以后照顾一下低层次读者，写好技术铺垫和名词定义。顺便问一下大家是怎么留言的，在手机上打那么多字，还有排版是怎么处理的，我是先在电脑上打好字再COPY上来的，大家和我一样吗?

2018-05-24

作者回复

非常感谢反馈;

关于引用计数，也有优势，我记得某个国外一线互联网公司调优python，就是只用引用计数，关闭gc

2018-05-28

小情绪 2

多谢杨老师，建议多结合代码讲解会更清晰，毕竟代码是最好的诠释。

2018-05-16

作者回复

好的，后面文章平衡下，贴多了代码也有人说凑字数...

2018-05-17

ASCE1885 2

Android开发面试中这道题大家基本都会，Java后端面试中遇到好些人说没听过的，说到底有部分人还是Java基础不过关，只会用框架。

2018-05-14

男人，7分熟。 2

留言区，个个都是人才

2018-05-14