**Java 引用详解**

从JDK1.2版本开始，把对象的引用分为四种级别，从而使程序能更加灵活的控制对象的生命周期。这四种级别由高到低依次为：强引用、软引用、弱引用和虚引用。

**1．强引用**  
本章前文介绍的引用实际上都是强引用，这是使用最普遍的引用。如果一个对象具有强引用，那就类似于必不可少的生活用品，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问题。

**2．软引用（SoftReference）**

如果一个对象只具有软引用，那就类似于可有可物的生活用品。如果内存空间足够，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。软引用可用来实现内存敏感的高速缓存。  
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，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。

**3．弱引用（WeakReference）**  
如果一个对象只具有弱引用，那就类似于可有可物的生活用品。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。  
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。

**4．虚引用（PhantomReference）**  
"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。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。  
虚引用主要用来跟踪对象被垃圾回收的活动。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。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。程序如果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。

在本书中，"引用"既可以作为动词，也可以作为名词，读者应该根据上下文来区分"引用"的含义。

在java.lang.ref包中提供了三个类：SoftReference类、WeakReference类和PhantomReference类，它 们分别代表软引用、弱引用和虚引用。ReferenceQueue类表示引用队列，它可以和这三种引用类联合使用，以便跟踪Java虚拟机回收所引用的对 象的活动。以下程序创建了一个String对象、ReferenceQueue对象和WeakReference对象：

//创建一个强引用  
String str = new String("hello");

//创建引用队列, <String>为范型标记，表明队列中存放String对象的引用  
ReferenceQueue<String> rq = new ReferenceQueue<String>();

//创建一个弱引用，它引用"hello"对象，并且与rq引用队列关联  
//<String>为范型标记，表明WeakReference会弱引用String对象  
WeakReference<String> wf = new WeakReference<String>(str, rq);

以上程序代码执行完毕，内存中引用与对象的关系如图11-10所示。

图11-10 "hello"对象同时具有强引用和弱引用

在图11-10中，带实线的箭头表示强引用，带虚线的箭头表示弱引用。从图中可以看出，此时"hello"对象被str强引用，并且被一个WeakReference对象弱引用，因此"hello"对象不会被垃圾回收。  
在以下程序代码中，把引用"hello"对象的str变量置为null，然后再通过WeakReference弱引用的get()方法获得"hello"对象的引用：

String str = new String("hello"); //①  
ReferenceQueue<String> rq = new ReferenceQueue<String>(); //②  
WeakReference<String> wf = new WeakReference<String>(str, rq); //③

str=null; //④取消"hello"对象的强引用  
String str1=wf.get(); //⑤假如"hello"对象没有被回收，str1引用"hello"对象

//假如"hello"对象没有被回收，rq.poll()返回null  
Reference<? extends String> ref=rq.poll(); //⑥

执行完以上第④行后，内存中引用与对象的关系如图11-11所示，此 时"hello"对象仅仅具有弱引用，因此它有可能被垃圾回收。假如它还没有被垃圾回收，那么接下来在第⑤行执行wf.get()方法会返 回"hello"对象的引用，并且使得这个对象被str1强引用。再接下来在第⑥行执行rq.poll()方法会返回null，因为此时引用队列中没有任 何引用。ReferenceQueue的poll()方法用于返回队列中的引用，如果没有则返回null。

图11-11 "hello"对象只具有弱引用

在以下程序代码中，执行完第④行后，"hello"对象仅仅具有弱引用。接下来两次调用System.gc()方法，催促垃圾回收器工作，从而提 高"hello"对象被回收的可能性。假如"hello"对象被回收，那么WeakReference对象的引用被加入到ReferenceQueue 中，接下来wf.get()方法返回null，并且rq.poll()方法返回WeakReference对象的引用。图11-12显示了执行完第⑧行后 内存中引用与对象的关系。

String str = new String("hello"); //①  
ReferenceQueue<String> rq = new ReferenceQueue<String>(); //②  
WeakReference<String> wf = new WeakReference<String>(str, rq); //③  
str=null; //④

//两次催促垃圾回收器工作，提高"hello"对象被回收的可能性  
System.gc(); //⑤  
System.gc(); //⑥  
String str1=wf.get(); //⑦ 假如"hello"对象被回收，str1为null  
Reference<? extends String> ref=rq.poll(); //⑧

图11-12 "hello"对象被垃圾回收，弱引用被加入到引用队列

The important part about strong references — the part that makes them "strong" — is how they interact with the garbage collector. Specifically, if an object is reachable via a chain of strong references (strongly reachable), it is not eligible for garbage collection. As you don’t want the garbage collector destroying objects you’re working on, this is normally exactly what you want.

<script type="text/javascript"></script>