强引用：

只要引用存在，垃圾回收器永远不会回收

Object obj = new Object();

//可直接通过obj取得对应的对象 如obj.equels(new Object());

而这样 obj对象对后面new Object的一个强引用，只有当obj这个引用被释放之后，对象才会被释放掉，这也是我们经常所用到的编码形式。

软引用：

非必须引用，内存溢出之前进行回收，可以通过以下代码实现

Object obj = new Object();

SoftReference<Object> sf = new SoftReference<Object>(obj);

obj = null;

sf.get();//有时候会返回null

这时候sf是对obj的一个软引用，通过sf.get()方法可以取到这个对象，当然，当这个对象被标记为需要回收的对象时，则返回null；

软引用主要用户实现类似缓存的功能。在内存足够的情况下直接通过软引用取值，无需从繁忙的真实来源查询数据，提升速度；当内存不足时，自动删除这部分缓存数据，从真正的来源查询这些数据。

弱引用：

第二次垃圾回收时回收，可以通过如下代码实现

Object obj = new Object();

WeakReference<Object> wf = new WeakReference<Object>(obj);

obj = null;

wf.get();//有时候会返回null

wf.isEnqueued();//返回是否被垃圾回收器标记为即将回收的垃圾

弱引用是在第二次垃圾回收时回收，短时间内通过弱引用取对应的数据，可以取到，当执行过第二次垃圾回收时，将返回null。

弱引用主要用于监控对象是否已经被垃圾回收器标记为即将回收的垃圾，可以通过弱引用的isEnQueued方法返回对象是否被垃圾回收器标记。

虚引用：

垃圾回收时回收，无法通过引用取到对象值，可以通过如下代码实现

Object obj = new Object();

PhantomReference<Object> pf = new PhantomReference<Object>(obj);

obj=null;

pf.get();//永远返回null

pf.isEnqueued();//返回是否从内存中已经删除

虚引用是每次垃圾回收的时候都会被回收，通过虚引用的get方法永远获取到的数据为null，因此也被成为幽灵引用。

虚引用主要用于检测对象是否已经从内存中删除。

最近在学习Java虚拟机，碰到引用的问题，在此借鉴总结一下：

原文地址：http://blog.csdn.net/coding_or_coded/article/details/6603549

对象的强、软、弱和虚引用

在JDK 1.2以前的版本中，若一个对象不被任何变量引用，那么程序就无法再使用这个对象。也就是说，只有对象处于可触及（reachable）状态，程序才能使用它。从JDK 1.2版本开始，把对象的引用分为4种级别，从而使程序能更加灵活地控制对象的生命周期。这4种级别由高到低依次为：强引用、软引用、弱引用和虚引用。

⑴**强引用（StrongReference）**

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。 ps：强引用其实也就是我们平时A a = new A()这个意思。

⑵**软引用（SoftReference）**

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。软引用可用来实现内存敏感的高速缓存（下文给出示例）。

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。

⑶**弱引用（WeakReference）**

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。

⑷**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器回收。

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。

使用软引用构建敏感数据的缓存

1 为什么需要使用软引用

首先，我们看一个雇员信息查询系统的实例。我们将使用一个Java语言实现的雇员信息查询系统查询存储在磁盘文件或者数据库中的雇员人事档案信息。作为一个用户，我们完全有可能需要回头去查看几分钟甚至几秒钟前查看过的雇员档案信息(同样，我们在浏览WEB页面的时候也经常会使用"后退"按钮)。这时我们通常会有两种程序实现方式：一种是把过去查看过的雇员信息保存在内存中，每一个存储了雇员档案信息的Java对象的生命周期贯穿整个应用程序始终；另一种是当用户开始查看其他雇员的档案信息的时候，把存储了当前所查看的雇员档案信息的Java对象结束引用，使得垃圾收集线程可以回收其所占用的内存空间，当用户再次需要浏览该雇员的档案信息的时候，重新构建该雇员的信息。很显然，第一种实现方法将造成大量的内存浪费，而第二种实现的缺陷在于即使垃圾收集线程还没有进行垃圾收集，包含雇员档案信息的对象仍然完好地保存在内存中，应用程序也要重新构建一个对象，我们知道，访问磁盘文件、访问网络资源、查询数据库等操作都是影响应用程序执行性能的重要因素，如果能重新获取那些尚未被回收的Java对象的引用，必将减少不必要的访问，大大提高程序的运行速度。

2 如果使用软引用

SoftReference的特点是它的一个实例保存对一个Java对象的软引用，该软引用的存在不妨碍垃圾收集线程对该Java对象的回收。也就是说，一旦SoftReference保存了对一个Java对象的软引用后，在垃圾线程对这个Java对象回收前，SoftReference类所提供的get()方法返回Java对象的强引用。另外，一旦垃圾线程回收该Java对象之后，get()方法将返回null。

看下面代码：
MyObject aRef = **new** MyObject();

SoftReference aSoftRef=**new** SoftReference(aRef);

此时，对于这个MyObject对象，有两个引用路径，一个是来自SoftReference对象的软引用，一个来自变量aReference的强引用，所以这个MyObject对象是强可及对象。

随即，我们可以结束aReference对这个MyObject实例的强引用：
aRef = **null**;

此后，这个MyObject对象成为了软可及对象。如果垃圾收集线程进行内存垃圾收集，并不会因为有一个SoftReference对该对象的引用而始终保留该对象。Java虚拟机的垃圾收集线程对软可及对象和其他一般Java对象进行了区别对待：软可及对象的清理是由垃圾收集线程根据其特定算法按照内存需求决定的。也就是说，垃圾收集线程会在虚拟机抛出OutOfMemoryError之前回收软可及对象，而且虚拟机会尽可能优先回收长时间闲置不用的软可及对象。对那些刚刚构建的或刚刚使用过的"新"软可反对象会被虚拟机尽可能保留。在回收这些对象之前，我们可以通过：

MyObject anotherRef=(MyObject)aSoftRef.get();

重新获得对该实例的强引用。而回收之后，调用get()方法就只能得到null了。

3 使用ReferenceQueue清除失去了软引用对象的SoftReference

作为一个Java对象，SoftReference对象除了具有保存软引用的特殊性之外，也具有Java对象的一般性。所以，当软可及对象被回收之后，虽然这个SoftReference对象的get()方法返回null,但这个SoftReference对象已经不再具有存在的价值，需要一个适当的清除机制，避免大量SoftReference对象带来的内存泄漏。在java.lang.ref包里还提供了ReferenceQueue。如果在创建SoftReference对象的时候，使用了一个ReferenceQueue对象作为参数提供给SoftReference的构造方法，如：
ReferenceQueue queue = **new** ReferenceQueue();

SoftReference ref=**new** SoftReference(aMyObject, queue);

那么当这个SoftReference所软引用的aMyObject被垃圾收集器回收的同时，ref所强引用的SoftReference对象被列入ReferenceQueue。也就是说，ReferenceQueue中保存的对象是Reference对象，而且是已经失去了它所软引用的对象的Reference对象。另外从ReferenceQueue这个名字也可以看出，它是一个队列，当我们调用它的poll()方法的时候，如果这个队列中不是空队列，那么将返回队列前面的那个Reference对象。

在任何时候，我们都可以调用ReferenceQueue的poll()方法来检查是否有它所关心的非强可及对象被回收。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，利用这个方法，我们可以检查哪个SoftReference所软引用的对象已经被回收。于是我们可以把这些失去所软引用的对象的SoftReference对象清除掉。常用的方式为：
SoftReference ref = **null**;

while ((ref = (EmployeeRef) q.poll()) != **null**) {

// 清除ref

}