**Java 持有引用 & WeakHashMap**

@author ixenos

* **Reference引用对象**

* + **可获得的（reachable）：**指在栈中有一个普通的引用正在指向此对象，也可能有更多的中间链接
    - 如果一个对象是可获得的，垃圾回收器（GC）就不能释放它，因为他仍然为程序所用，宁愿耗尽内存也不释放；当一个对象不是可获得的时候，程序无法使用它，此时GC才认为将其回收是安全的
  + **Reference对象：**如果想继续持有对某个对象的引用，但也希望能够允许GC释放它，就应使用Reference对象作为与普通引用之间的媒介（代理），即普通引用经Reference对象包装过再指向某个对象
  + **普通引用-->Reference对象-->目标对象**

* **Reference引用对象的四种分类（由强到弱）**

* + 强引用（Strong Reference）：强引用时，就是普通引用，对象是**可获得的（reachable）**
  + 软引用（Soft Reference）：实现内存敏感的高速缓存 //TODO
  + **弱引用（Weak Reference）**：实现**规范映射（canonicalizing mappings）**，它不妨碍GC回收映射的key或value，**规范映射**中对象的实例可以在程序的多处被同时使用，以节省存储空间
  + 幻象引用（Phantom Reference）：用以调度回收前的清理工作，比Java终止机制灵活

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 幻象引用原理

* **ReferenceQueue引用队列**
  + 使用**软引用**和**弱引用**时们可以选择是否要将他们放入ReferenceQueue（用作“回收前清理工作”的工具），而**幻象引用**回收后进入ReferenceQueue
  + **幻象引用**和**弱引用**的不同在于其入队（enquene）进入ReferenceQuene的方式：
    - 当弱引用的对象成为若可到达时，弱引用即列队进入ReferenceQuene。这个入队发生在终结（finialize）和垃圾回收实际发生之前。理论上，通过不正规（unorthodox）的finilize（）方法，成为垃圾的对象能重新复活（resurrected），但是弱引用仍然是死的，因为没有引用指向它；
    - 而幻象引用只有当对象在物理上从内存中移出时，才会入队。这就阻止我们重新恢复将死的对象；
    - 终结(Finalization)指比拉圾回收更一般的概念,可以回收对象所占有的任意资源,比如文件描述符和图形上下文等
  + 以**弱引用**为例
    - 一旦弱引用返回null值，那么其指向的对象就变成了垃圾，这个弱引用对象也就没有用了。这通常意味着要进行一定方式的清理（cleanup）。例如，WeakHashmap将会移除一些死的（dread）的entry，避免持有过多死的弱引用。ReferenceQuene能够轻易的**追踪这些死掉的弱引用；**
    - 可以将ReferenceQuene传入WeakHashmap的构造方法（constructor）中，这样，**一旦这个弱引用指向的对象成为垃圾，这个弱引用将加入ReferenceQuene中**

* **WeakHashMap弱散列映射表**
  + WeakHashMap使用弱引用（WeakReference）保存键，称为弱键
  + ref-->weak ref-->key ,  if (ref==null) , than ( weak ref-->ref queue , WeakHashMap将删除对应的条目)
  + **动态回收**的步骤：

(1) **新建WeakHashMap，将“键值对”添加到WeakHashMap中**

        将“键值对”添加到WeakHashMap中时，添加的键都是弱键；

        实际上，WeakHashMap是通过数组table保存Entry(键值对)；每一个Entry实际上是一个单向链表，即Entry是键值对链表；

(2) **当某“弱键”不再被其它对象引用，并被GC回收时。在GC回收该“弱键”时，这个“弱键”也同时会被添加到queue队列中**

        例如，当我们在将“弱键”key添加到WeakHashMap之后；后来将key设为null。这时，便没有外部外部对象再引用该了key；

        接着，当Java虚拟机的GC回收内存时，会回收key的相关内存；同时，将key添加到queue队列中；

(3)**当下一次我们需要操作WeakHashMap时，会先同步table和queue。table中保存了全部的键值对，而queue中保存被GC回收的“弱键”；同步它们，就是删除table中被GC回收的“弱键”对应的键值对**

        例如，当我们“读取WeakHashMap中的元素或获取WeakReference的大小时”，它会先同步table和queue，目的是“删除table中被GC回收的‘弱键’对应的键值对”。删除的方法就是逐个比较“table中元素的‘键’和queue中的‘键’”，若它们相当，则删除“table中的该键值对”。