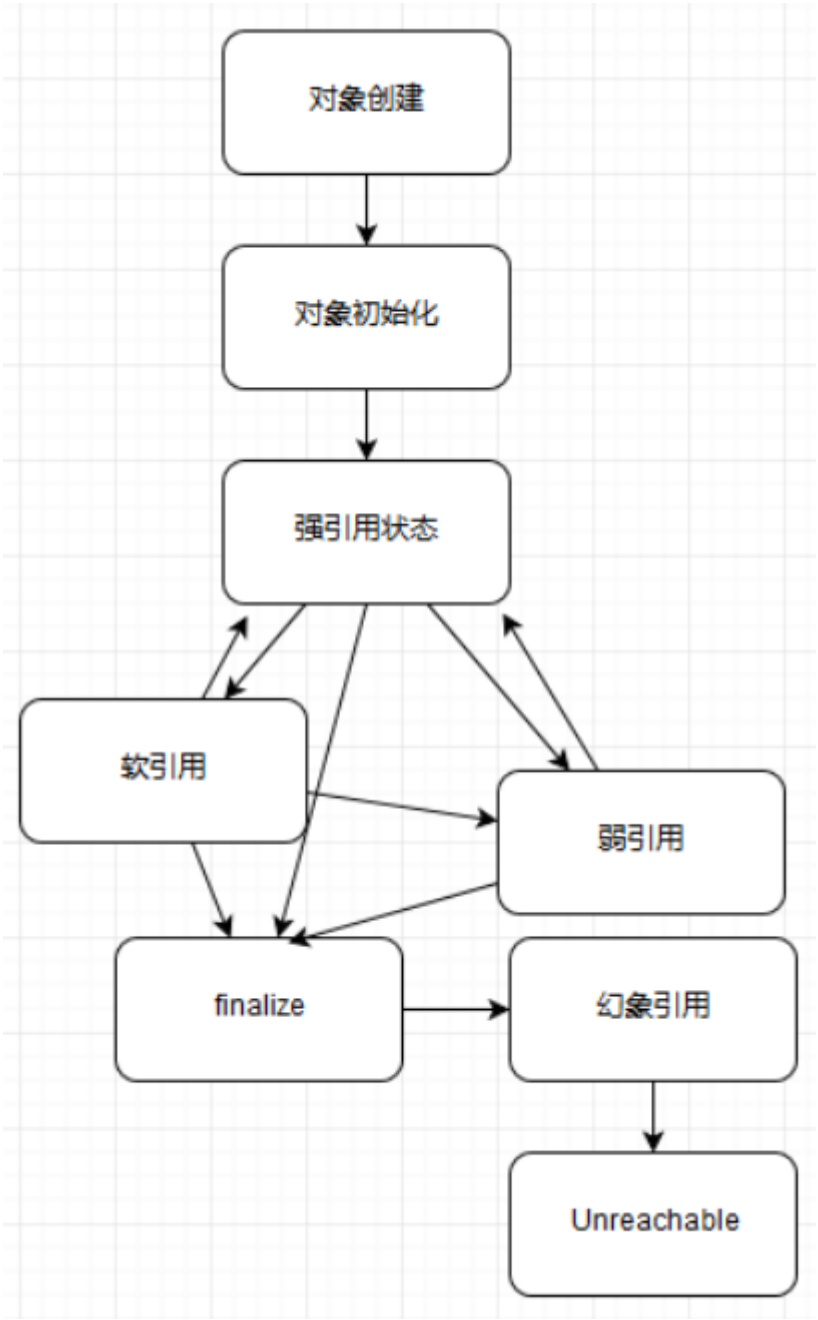
- **引用出现的根源**由于GC **内存回收本质上是回收对象**

  对象可达性状态流转分析（对象生命周期和不同可达状态）



强引用：常见的普通对象引用，只要还有强引用指向一个对象，就表明对象"活着"， 垃圾收集器不会碰这种对象。 对于一个普通的对象，如果没有其它引用关系，只要超出引用作用域或者显示的赋值为null, 就可以被垃圾收集了，当然具体回收时机还是要看垃圾收集策略。

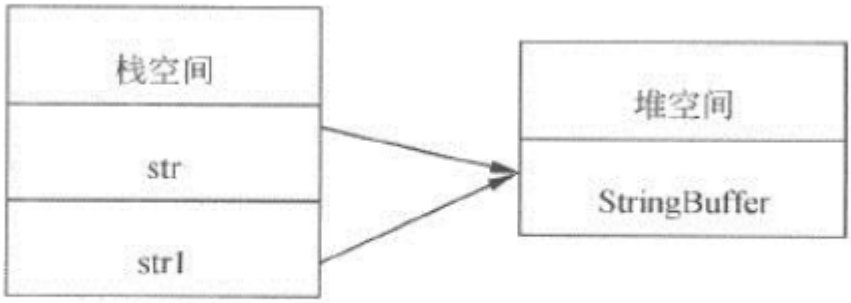


图 3.25　强引用示例 2

软引用：通过SoftReference 可以让JVM 避免一些垃圾收集，只有JVM 认为内存不足了， 才会试图去回收对象。

比如在OutMemoryError 之前会清理软引用。

弱引用： 仅仅提供了一种访问在弱引用对象的途径， 不能避免被垃圾收集。 比如如果对象还在就使用它，否则重新实例化。

幻象引用: 虚引用，不能通过它访问对象， 它仅仅提供了一种确保对象被finalize 以后， 做某些事情的机制。比如Cleaner

这是java 定义了不同可达性级别：

\1. 强可达： 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。

（比如新创建一个对象，那么创建它的线程对它就是强可达）

\2. 软可达： 只能通过软引用才能访问到对象状态

\3. 弱可达： 只能通过弱引用访问时的状态， 十分临近finalize 状态的时机

\4. 幻象可达： 没有强，软，弱引用关联，并且fialize 过了，只有幻象引用指向这个对象。

\5. 不可达： 可以被清除了

所有引用类型，都是抽象类java.lang.ref.Reference 的子类，可以人为的将非幻象可达的对象改为强可达。 检测弱引用对象是否被垃圾收集，也是诊断内存泄漏的一个思路。

引用队列的使用：

创建各种关联对象的时侯，可以选择是否需要关联引用队列。可以从队列里获取引用。

可以通过java 提供的ReferenceQueue 和 PhantomReference 对对象进行保存

软引用会在最后一次引用后， 还能保存一段时间，可以通过参数指定。

\4. 诊断JVM 引用情况

工具： 比如HotSpot JVM 提供的选项PrintReferebceGC 、PrintGCTimeStamps

\5. Reachability Fence

通过底层API 来达到强引用效果，避免被意外回收， 比如在finally 包围起来，明确声明对象强可达。