复执行。由于是在脚本末尾递归调用，因此这种形式的递归被称为尾部递归。当然，你也可以在之前递归调用，但本书并不对这种递归方式做详细讨论。

在实践中，无限的递归并不是很常用。递归通常需要使用条件语句进行控制，Scratch 使用**如果…那么**等分支结构控制递归。图 7-18 展示了一个递归的过程，它从某个指定的数字（由参数 count 指定）减少到 0。

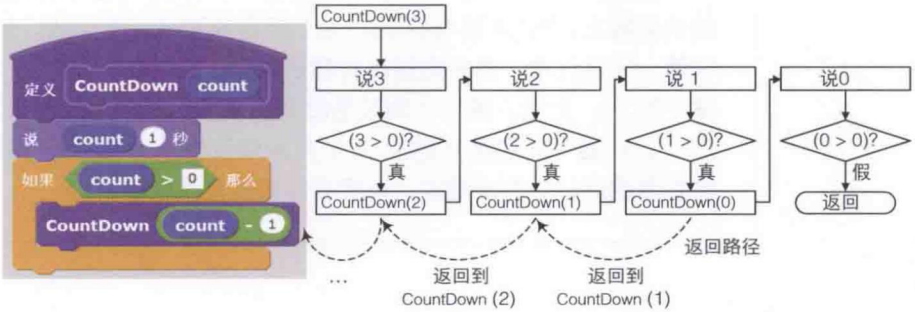

图 7-18：使用如果…那么积木控制递归的执行

当以参数 3 调用过程 CountDown 时，递归是如何进行的呢？过程开始时，**说…**展示数字 3，然后检查 count 是否大于 0。因为 3 大于 0，故以参数 2（count-1）调用过程自身。

第二次调用时，过程首先展示数字 2。因为数字 2 大于 0，它会再一次以参数 1 调用自身。依次重复直到调用了 CountDown(0)。在展示了数字 0 后，过程将检查变量 count 是否大于 0。因为该布尔表达式的结果为假，递归将终止，过程依次返回。尝试跟着图 7-18 的流程走一遍。

我们可以使用递归制作许多好玩且有趣的程序，例如，图 7-19 的过程 Blade。

角色的起始方向是 90°。当绘制一个正三角形后，角色移动 12 步，并**逆时针旋转 10°**。随后过程检查角色的新方向：如果角色未面向 90°，过程则重新调用自己，继续绘制下一个正三角形；否则递归结束，过程停止。最终的绘制结果如图 7-19 所示。