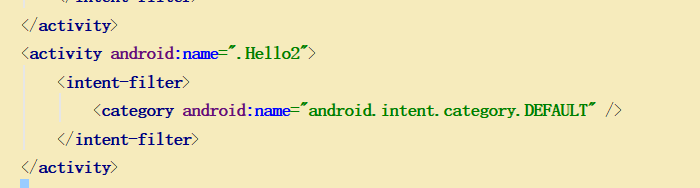
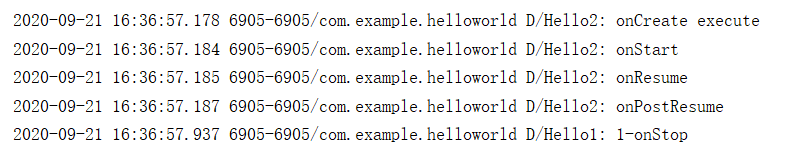
实验二 4种启动模式

standard模式：

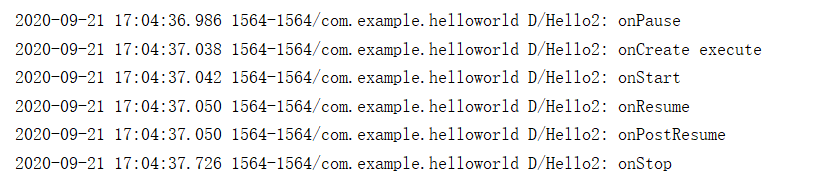
启动模式为standard时，系统不会在乎这个活动是否已经在返回栈中存在，每次启动都会创建该活动的一个新的实例。

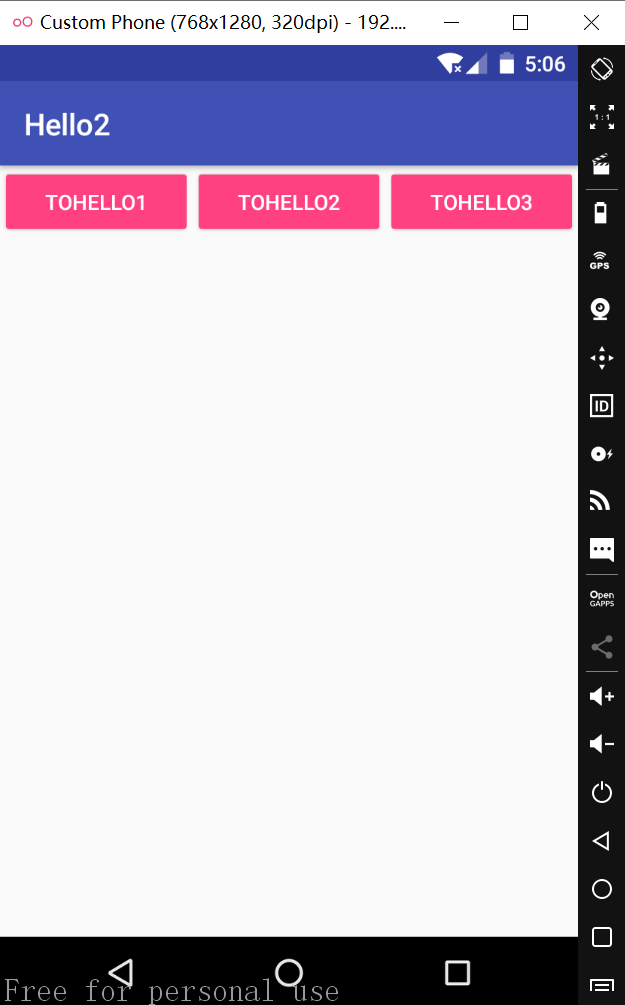


Hello2的启动模式就是standard模式，点击Hello2，日志信息如下：

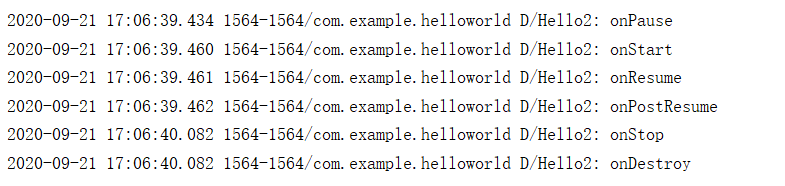


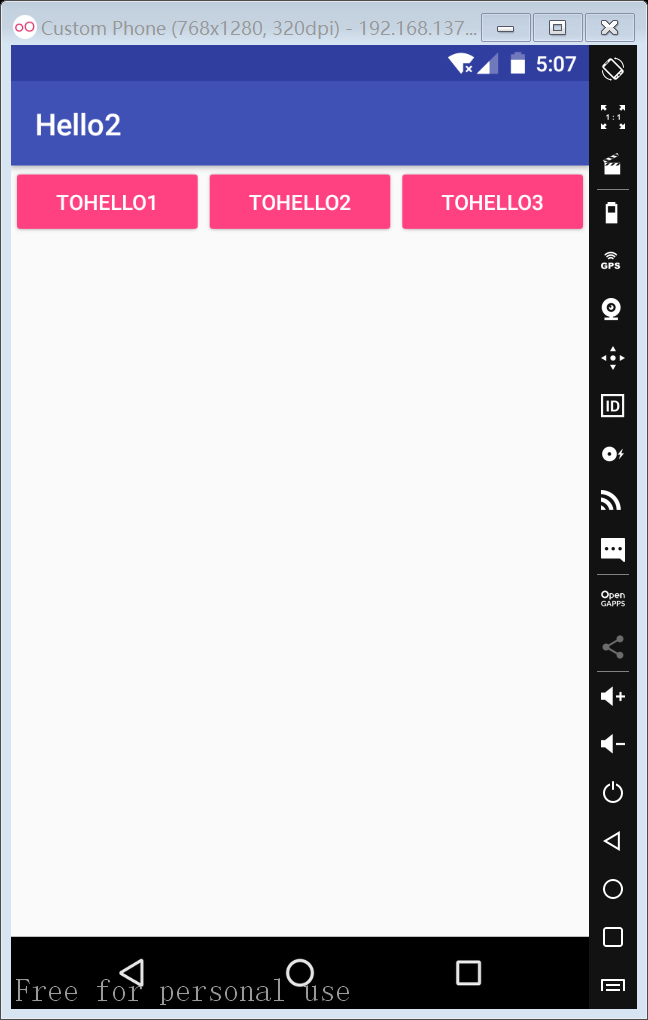
此时再此点击Hello2，可以看出此时的Hello2又调用了onCreate()创建了一个新的实例。



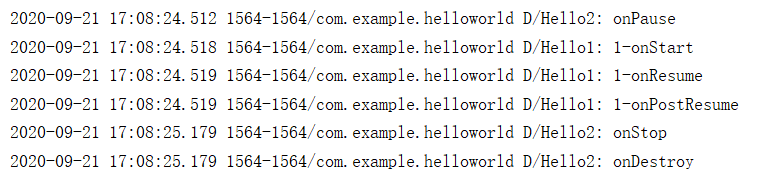


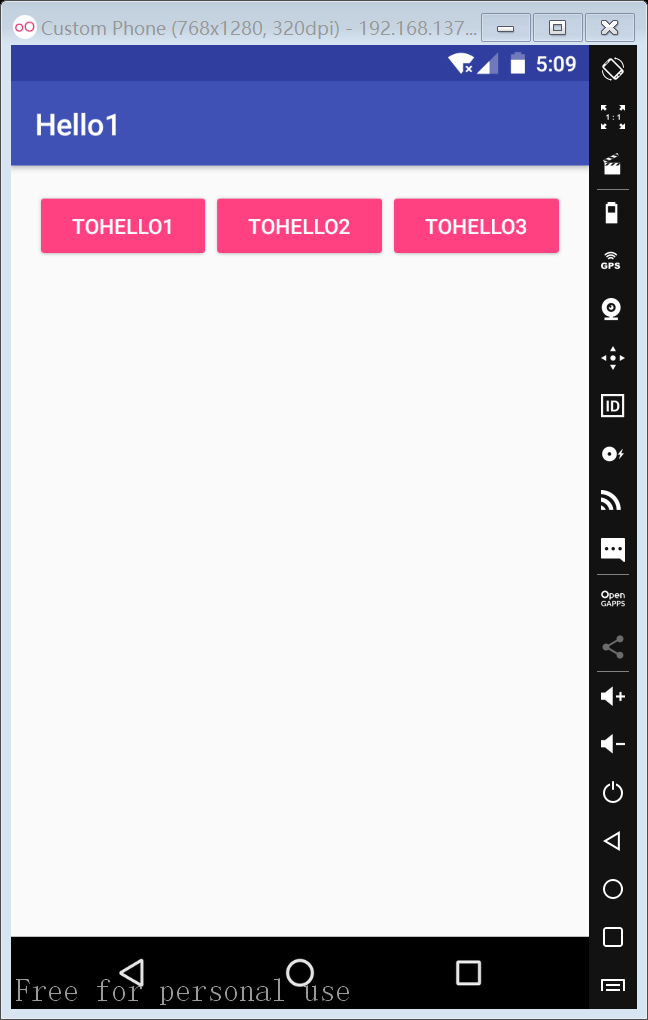
点击back键后，执行onResume(),onStop()后onDestroy(),销毁了当前这个实例。此时界面还是如下：





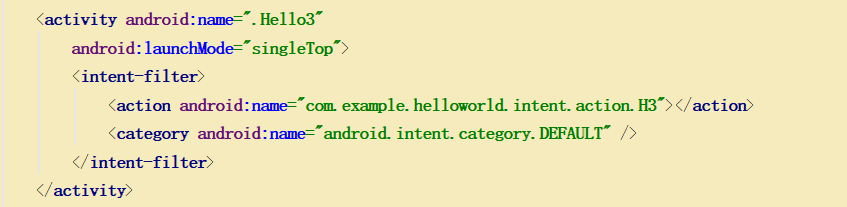
再次点击back键，才回到Hello1,而Hello2的这一个实例执行onStop()，onDestroy()后,销毁了这个实例。



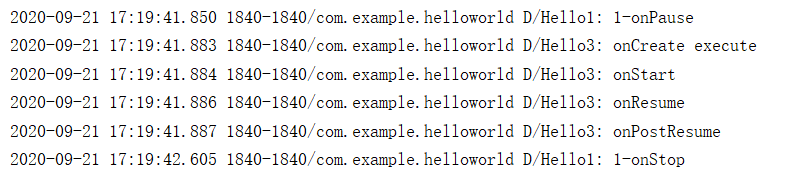


singleTop模式：

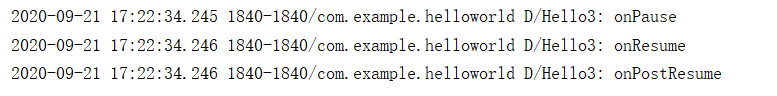
启动方式singleTop时，启动活动时如果发现返回栈的栈顶已经是该活动，则认为可以直接使用它，不会再创建新的活动实例。



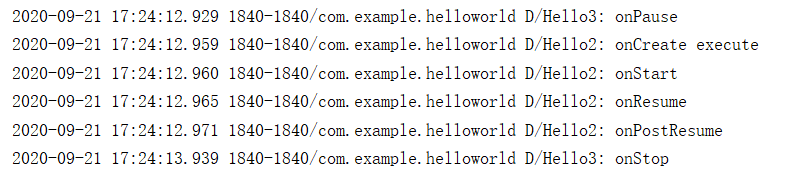
此时Hello3 singleTop模式，使用点击Hello3：



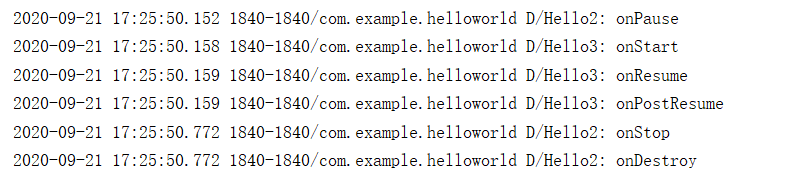
再此点击Hello3，可以看出此时的Hello3只是调用了onPause(),onResume()，没有调用onCreate()，说明没有创建新的实例。



点击Hello2：



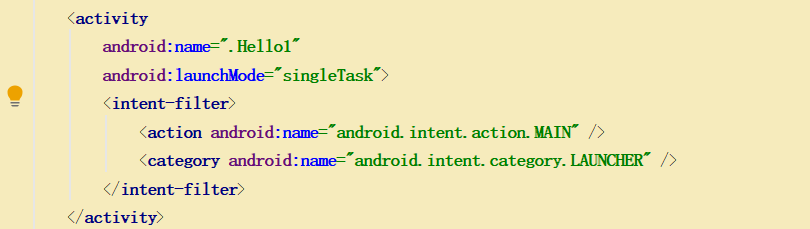
此时Hello3调用了onStop()，点击返回键：



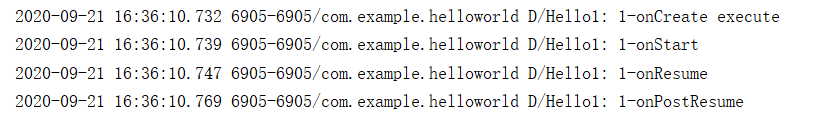
Hello3执行onStart()变为可见，Hello2活动执行onStop()和onDestroy()后被销毁。因此可以看出当singleTop的启动模式处于栈顶时，不会创建新的实例。

singleTask模式：

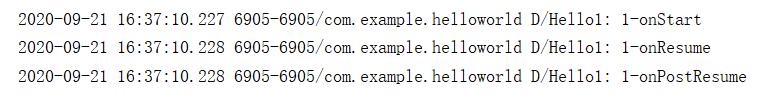
当活动的启动模式指定为singleTask，每次启动该活动时系统首先会在返回栈中检查是否存在该活动的实例，如果发现已经存在则直接使用该实例，并把在这个活动之上的所有活动统统出栈，如果没有发现就会创建一个新的活动实例。



Hello1的启动模式singleTask，此时运行程序如下图：

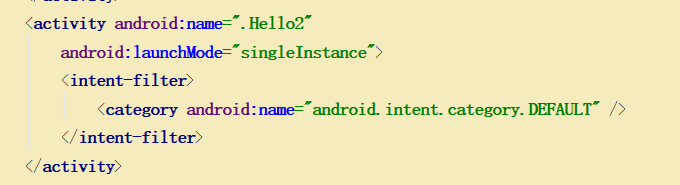


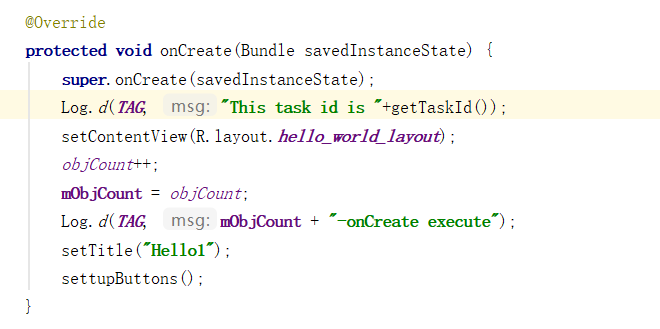
再此点击Hello1，可以看出再次点击Hello1，活动并没有调用onCreate()，而是回到它原本的实例，调用onStart()。

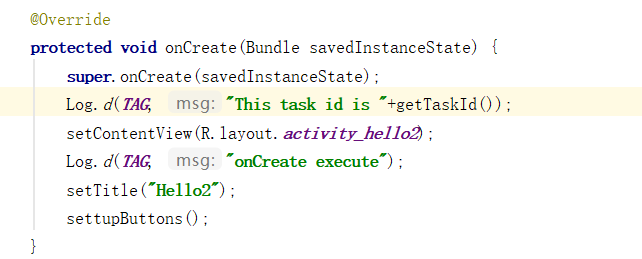


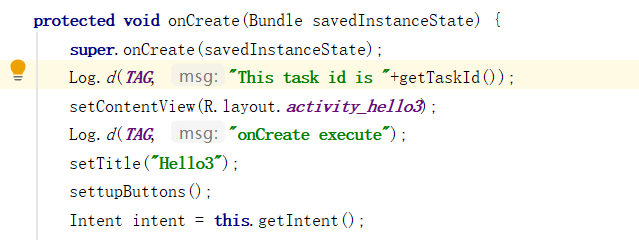
singleInstance模式

启动模式是singleInstance时，会有一个单独的返回栈来管理这个活动，不管是哪个应用程序来访问这个活动，都公用同一个返回栈，也就可以解决了共享活动实例的问题。









在程序中运行Hello1(),Hello2(),Hello3(),查看日志里的信息。在Hello1中，返回栈的id为26，在运行Hello2时，返回栈的id变成了27，而关闭Hello2,运行Hello3时，返回栈的id也是26，说明Hello1和Hello3是同一个返回栈。

