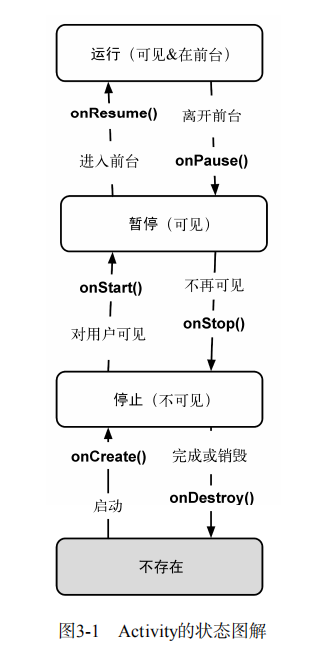
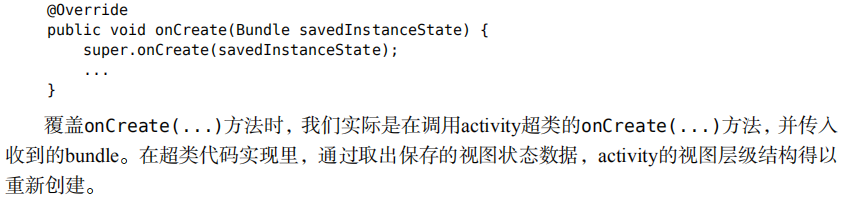


activity在运行、暂停和停止三种可能的状态间进行转换。每次状态发生转换时，都有对应的Activity方法将状态改变的消息通知activity。



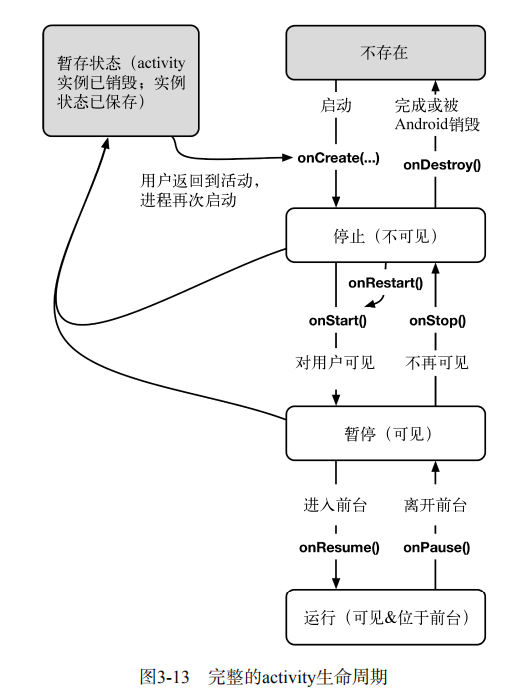
设备一经旋转，Android需要销毁当前的activity，然后新建一个activity来完成onCreate（）方法的调用，从而实现使用最佳资源匹配新的设备。

设备旋转前保存数据：



可通过覆盖onSaveInstanceState()方法，将一些数据保存在bundle中，然后在onCreate（）方法中取回这些数据。

基于用户体验考虑，Android从不会为了回收内存，而去销毁正在运行的activity。activity只有在暂停或停止状态下才可能会被销毁。此时会调用onSaveInstanceState()将用户数据保存在Bundle对象中。然后操作系统将Bundle对象放入activity记录中。



有时，Android不仅会销毁activity，还会彻底停止当前应用的进程。不过，只有在用户离开当前应用时才会发生这种情况。即使这种情况真的发生了，暂存的activity记录依然被系统保留着，以便于用户返回时activity的快速恢复。

使用startActivityForResult(Intent intent, int requestCode)方法打开新的Activity，我们需要为startActivityForResult()方法传入一个请求码(第二个参数)。请求码的值是根据业务需要由自已设定，用于标识请求来源。例如：一个Activity有两个按钮，点击这两个按钮都会打开同一个Activity，不管是那个按钮打开新Activity，当这个新Activity关闭后，系统都会调用前面Activity的onActivityResult(int requestCode, int resultCode, Intent data)方法。在onActivityResult()方法如果需要知道新Activity是由那个按钮打开的，并且要做出相应的业务处理，这时可以这样做：

 public void onCreate(Bundle savedInstanceState) {

            ....

        button1.setOnClickListener(new View.OnClickListener(){

            public void onClick(View v) {

                startActivityForResult (new Intent(MainActivity.this, NewActivity.class), 1);

            }

        });

        button2.setOnClickListener(new View.OnClickListener(){

            public void onClick(View v) {

             startActivityForResult (new Intent(MainActivity.this, NewActivity.class), 2);

            }

        });

        @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {

            switch(requestCode){

                case 1:

                //来自按钮1的请求，作相应业务处理

                case 2:

                //来自按钮2的请求，作相应业务处理

            }

        }

    }

三.结果码的作用

    在一个Activity中，可能会使用startActivityForResult()方法打开多个不同的Activity处理不同的业务，当这些新Activity关闭后，系统都会调用前面Activity的onActivityResult(int requestCode, int resultCode, Intent data)方法。为了知道返回的数据来自于哪个新Activity，在onActivityResult()方法中可以这样做(ResultActivity和NewActivity为要打开的新Activity)：

public class ResultActivity extends Activity {

        .....

        ResultActivity.this.setResult(1, intent);

        ResultActivity.this.finish();

    }

    public class NewActivity extends Activity {

        ......

        NewActivity.this.setResult(2, intent);

        NewActivity.this.finish();

    }

    public class MainActivity extends Activity { // 在该Activity会打开ResultActivity和NewActivity

        @Override

        protected void onActivityResult(int requestCode, int resultCode, Intent data) {

            switch(resultCode){

                case 1:

                // ResultActivity的返回数据

                case 2:

                // NewActivity的返回数据

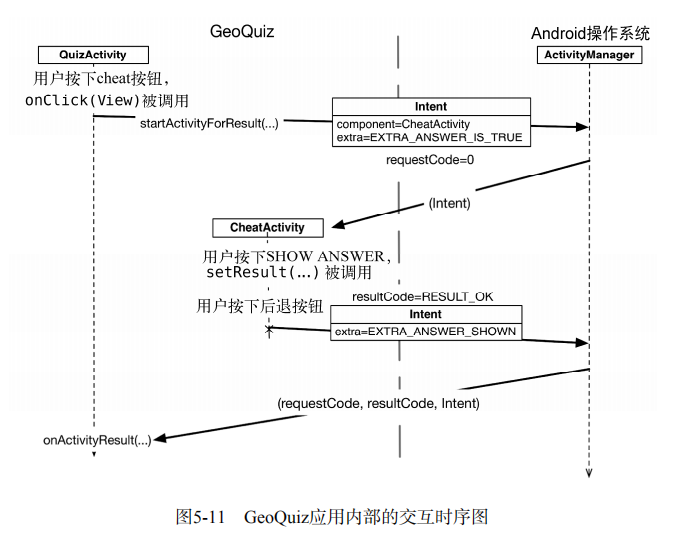
            }

        }

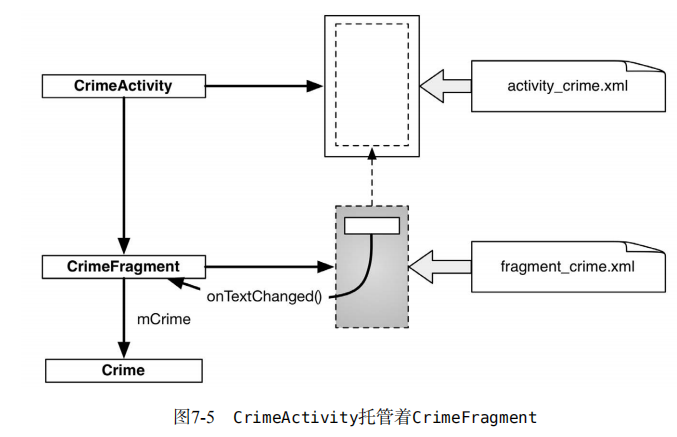
    }

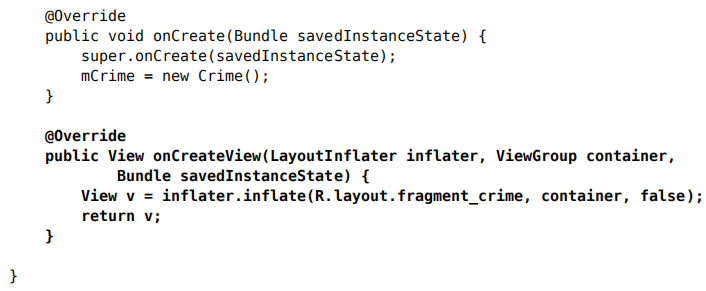
上面的栗子说明，requestCode 和 resultCode都是返回提出请求的MainActivity结果处理函数onActivityResult()使用的标识，只是requestCode 主要用来标识是MainActivity也就是请求Activity发出的哪个请求，而resultCode则是标识哪个Activity处理并返回的请求。

但是，无论requestCode 还是 resultCode都无法用于处理请求的Activity标识是谁发出的请求。因此需要自行传递参数到处理请求的Activity。



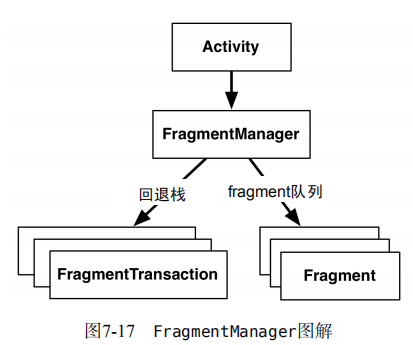
Activity在其视图层里提供一处位置用来放置fragment的视图，fragment本身不具备在屏幕上显示视图的能力。因此，只有将它的视图放置在activity的视图层级结构中,fragment视图才能显示在屏幕上。

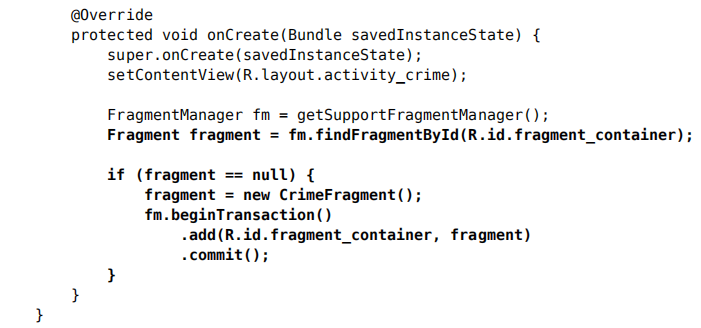




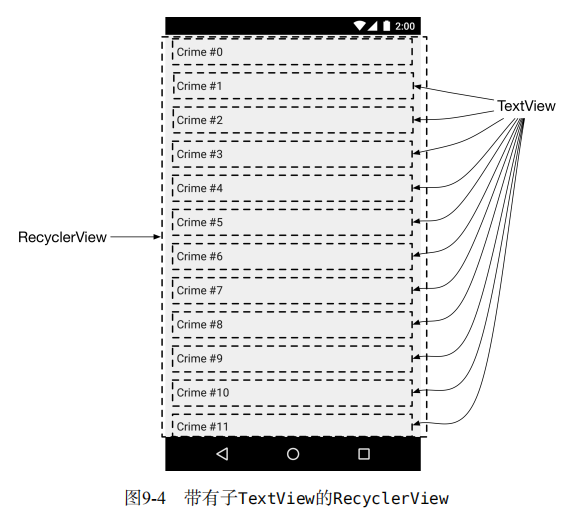
onCreateView()方法实例化fragment视图的布局，然后将实例化的view返回给托管activity。Bundle用来存储恢复数据，可供该方法从保存状态下重建视图。

在该方法中，fragment 的视图是直接通过LayoutInflater.inflate()方法并传入布局的资源ID生成的。第二个参数是视图的父布局，我们通常需要父视图来正确配置组件。第三个参数告知布局生成器是否将生成的视图添加给父布局。这里，我们传入了false参数，因此我们将以activity代码的方式添加视图。

FragmentManager类负责管理fragment并将它们的视图添加到activity的视图层级结构中。

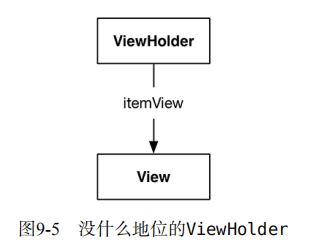


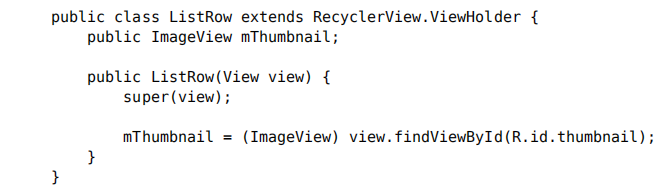
Fragment事务被用来添加、移除、附加、分离或替换fragment队列中的fragment。FragmentManager管理着fragment事务回退栈。



RecyclerView:它只创建刚好充满屏幕的12个视图，而不是100个视图。用户滑动屏幕切换视图时，上一个视图会被回收利用。RecyclerView所做的就是回收再利用，循环往复。

ViewHolder只做一件事：容纳View视图



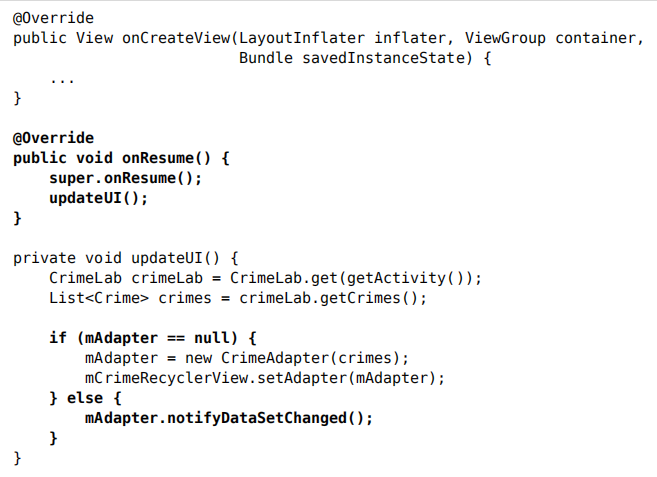


Adapter：RecyclerView自己不创建ViewHolder。这个任务实际是由adapter来完成的。adapter是个控制器对象，从模型层获取数据，然后提供给RecyclerView显示，起到了沟通的桥梁作用。

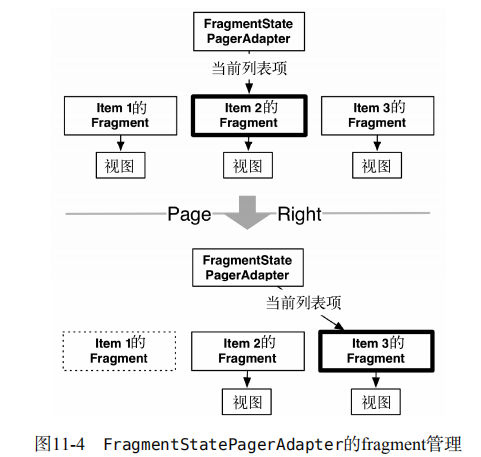
Adapter负责：

1. 创建必要的ViewHolder;
2. 绑定ViewHolder至模型层数据。

刷新显示列表项



FragmentStatePagerAdapter会销毁不需要的fragment。事务提交后，activity的FragmentManager中的fragment会被彻底移除。FragmentStatePagerAdapter类名中“state”表明：在销毁fragment时，可在onSaveInstanceState(Bundle)方法中保fragment的Bundle信息。用户切换回来时，保存的实例状态可用来恢复生成新的fragment



相比之下，FragmentPagerAdapter有不同的做法。对于不再需要的fragment，Fragment

PagerAdapter会选择调用事务的detach(Fragment)方法来处理它，而非remove(Fragment)方 法。也就是说，FragmentPagerAdapter只是销毁了fragment的视图，fragment实例还保留在 FragmentManager中。因此，FragmentPagerAdapter创建的fragment永远不会被销毁。

