## Activity的内容

建立Activity、在manifest.xml配置Activity

打开、关闭Activity

利用Bundle对不同Activity之间的数据交换

activity的回调机制

Activity的生命周期

## 1、什么是Android Application？

简单来说，一个apk文件就是一个Application。

任何一个android Application基本上是由一些Activities组成，当用户与应用程序交互时其所包含的部分Activities具有紧密的逻辑关系，或者各自独立处理不同的响应。

这些Activities捆绑在一起成为了一个处理特定需求的Application, 并且以“.apk”作为后缀名存在于文件系统中。

Android平台默认下的应用程序，例如：Email、Calendar、Browser、Maps、Text Message、Contacts、Camera和Dialer等都是一个个独立的Apps。

安装 Application的过程也可以简单理解为将其所包裹的Activities导入到当前的系统中，如果系统中已经存在了相同的Activities， 那么将会自动将其关联，而不会重复安装相同的Activities，避免资源的浪费。（是如何比较两个Activity是相同的呢？？？）

Application卸载的过程也会检查当前所关联的，Activities是否有被其它Application标签所关联，如果仅仅是提供当前的Application使用，那么将会彻底被移除，相反则不做任何操作。

## 2、什么是 Activity Stack？

操作应用程序时，有时需要调用多个Activities来完成需求，例如：发送邮件程序，首先是进入邮件主界面，然后启动一个新的Activity用于填写新邮件内容，同时可以调出联系人列表用于插入收件人信息等等。在这个操作过程中 Android平台有一个专门用于管理Activities堆栈的机制，其可以方便的线性记录Activities实例，当完成某个操作时，可以通过导航功能返回之前的Activity（通过按操作台的“Back”按钮）。

每次启动新的Activity都将被添加到Activity Stack。用户可以方便的返回上一个Activity直到Home Screen，到达Home Screen后，将无法再继续查看堆栈记录（俗话说：到头了）。如果当前Task被中止(Interrupting the task)，返回到系统主界面后启动了其它操作，当希望返回到前一个Task继续执行时，只需要再次通过主界面的Application launcher或者快捷方式启动这个Task的Root Activity便可返回其中止时的状态继续执行。

相对于Views、Windows、Menus和Dialogs而言，Activity是唯一可被记录在History stack中的数据，所以当你所设计的应用程序需要用户由A界面进入到次一级界面B，当完成操作后需要再次返回A,那么必须考虑将A看作为 Activity，否则将无法从历史堆栈中返回。

## 3、什么是Task？

当我们需要一个Activity可以启动另一个Activity，可能另外一个Activity是定义在不同应用程序中的Activity。

例如，假设你想在你的应用中让用户显示一些地方的街景。而这里已经有一个Activity可以做到这一点，因此，你的Activity所需要做的只是在Intent对象中添加必要的信息，并传递给startActivity()。地图浏览将会显示你的地图。当用户按下BACK键，你的Activity会再次出现在屏幕上。

对于用户来说，看起来好像是地图浏览与你的Activity一样，属于相同的应用程序，即便是它定义在其它的应用程序里，并运行在那个应用程序的进程里。

Android通过将这两个Activity保存在同一个Task里来体现这一用户体验。在一个Task堆栈里的活动Activity可以来自同一个应用程序也可以是来自不同的应用。简单来说，一个Task就是用户体验上的一个“应用”。 Task没有实体的堆栈数据结构，它只是逻辑上的一个堆栈。它将相关的Activity组合在一起，以stack的方式管理（就是前面提到的Activity Stack），这就是Task。

## 4、Task 有啥用？

我们用过Android的手机就会知道有下面的场景：

假设我们首先在用IReader在看书，从选书到具体书的阅读界面，这是有好几个Activity。我们每一个点击的Activity都被放在阅读这个Task对应的Activity Stack中了，这可以放我们通过回退键返回每一个前面的Activity。

我们在阅读到一半时，想看看Sina微博，按Home键离开了IReader。

在Sina微博界面也是有多个Activity，我们一步到阅读界面。这时候我们每一个点击的Activity都被放在Sina微博这个Task对应的Activity Stack中了，这可以放我们通过回退键返回每一个前面的Activity。

我们这时候再回到IReader读书界面，原先的状态还是保留的。

显然每一个Task有自己的 Activity Stack。

Task就是这样为了方便人们使用手机而设置的，就像前面提到的场景Task可以跨Application。

下面这个图从另外一个角度描述了Application Task Activities的关系。

## 5、任务栈是什么

任务栈Task，是一种用来放置Activity实例的容器，他是以栈的形式进行盛放，也就是所谓的先进后出，主要有2个基本操作：压栈和出栈，其所存放的Activity是不支持重新排序的，只能根据压栈和出栈操作更改Activity的顺序。

启动一个Application的时候，系统会为它默认创建一个对应的Task，用来放置根Activity。默认启动Activity会放在同一个Task中，新启动的Activity会被压入启动它的那个Activity的栈中，并且显示它。当用户按下回退键时，这个Activity就会被弹出栈，按下Home键回到桌面，再启动另一个应用，这时候之前那个Task就被移到后台，成为后台任务栈，而刚启动的那个Task就被调到前台，成为前台任务栈，Android系统显示的就是前台任务栈中的Top实例Activity。

## 6、任务栈的作用

以往基于应用（application）的程序开发中，程序具有明确的边界，一个程序就是一个应用，一个应用为了实现功能可以采用开辟新线程甚至新进程来辅助，但是应用与应用之间不能复用资源和功能。

而Android引入了基于组件开发的软件架构，虽然我们开发android程序，仍然使用一个apk工程一个Application的开发形式，但是对于Aplication的开发就用到了Activity、service等四大组件，其中的每一个组件，都是可以被跨应用复用的，这就是android的神奇之处。

虽然组件可以跨应用被调用，但是一个组件所在的进程必须是在组件所在的Aplication进程中。由于android强化了组件概念，弱化了Aplication的概念，所以在android程序开发中，A应用的A组件想要使用拍照或录像的功能就可以不用去针对Camera类进行开发，直接调用系统自带的摄像头应用（称其B应用）中的组件（称其B组件）就可以了，

但是这就引发了一个新问题，A组件运行在A应用中，B组件运行在B应用中，自然都不在同一个进程中，那么从B组件中返回的时候，如何实现正确返回到A组件呢？Task就是来负责实现这个功能的，它是从用户角度来理解应用而建立的一个抽象概念。因为用户所能看到的组件就是Activity，所以Task可以理解为实现一个功能而负责管理所有用到的Activit

栈是一个先进后出的线性表，根据Activity在当前栈结构中的位置，来决定该Activity的状态。正常情况下，当一个Activity启动了另一个Activity的时候，新启动的Activity就会置于任务栈的顶端，并处于活动状态，而启动它的Activity虽然成功身退，但依然保留在任务栈中，处于停止状态，当用户按下返回键或者调用finish()方法时，系统会移除顶部Activity，让后面的Activity恢复活动状态。当然，世界不可能一直这么“和谐”，可以给Activity设置一些“特权”，来打破这种“和谐”的模式，这种特权，就是通过在AndroidManifest文件中的属性andorid:launchMode来设置或者通过Intent的flag来设置的，下面就先介绍下Activity的几种启动模式。

## 7、打开Activity 活动

（1）打开一个新的Activity，不传递参数

public void OpenNew(View v){

//新建一个显式意图，第一个参数为当前Activity类对象，第二个参数为你要打开的 //Activity类

Intent intent =new Intent(MainActivity.this,MainActivity2.class);

startActivity(intent);

}

（2）打开新的Activity并传递参数

MainActivity.java传递参数

public void OpenNew(View v){

//新建一个显式意图，第一个参数为当前Activity类对象，第二个参数为你要打开的 //Activity类

Intent intent =new Intent(MainActivity.this,MainActivity2.class);

//用Bundle携带数据

Bundle bundle=new Bundle();

//传递name参数为tinyphp

bundle.putString("name", "tinyphp");

intent.putExtras(bundle);

startActivity(intent);

}

MainActivity2.java接收参数

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.newtest);

//新页面接收数据

Bundle bundle = this.getIntent().getExtras();

//接收name值

String name = bundle.getString("name");

Log.i("获取到的name值为",name);

}

（3）requestCode 请求码，即调用startActivityForResult() 传递过去的；

resultCode 结果码，结果码用于标识返回数据来自哪个新Activity。

1. 上面几种都是通过指定具体的Activity来启动的，为显示启动。

还有一类是隐式启动：

隐式启动：通过Intent-filter的Action动作功能，Category类别来实现，这个是通过Intent的 intent-filter来实现的。

## 8、优化加载Activity页面的操作

降低onCreate， onResume里的处理耗时；

降低界面没显示出来时的主线程占用；

避免在主线程调用阻塞方法；

简化布局，减少资源加载和布局计算的时间。

## 9、Activity之间的传值

1. 通过startActivity来进行Activity的传值；
2. 通过startActivityForResult方法来得到Activity的回传值，重写onActivityResult()；
3. 先把数据存在本地，然后在下一个Activity中从本地取;
4. SharedPreferences传递数据；SharedPrefences实际上是在本地创建一个xml文件。注意操作时线程阻塞。
5. 使用EventBus框架.

越来越多的项目开发中使用到EventBus这个框架，那什么是EventBus呢？用github开源文档上的介绍就是：一个Android平台的事件总线框架, 它简化了Activity、Fragment、Service等组件之间的交互，很大程度上降低了它们之间的耦合，使得我们的代码更加简洁，耦合性更低，提升我们的代码质量。

开发中我们都知道，很多时候要在Activty与Activty间，Activty与Fragment等之间传递数据进行交互，操作起来很不方便，我们以前可能会用到接口回调，或是用观察者模式来实现，或是发广播等等。这们操作不仅仅代码量大，而且代码耦合性高，性能不佳，也不便于维护。例如：两个Fragment之间的通信你会怎么实现？ 按照Android官方给的建议的解决方法如下: Communicating with the Activity,思路就是Activity实现某个接口,然后在Fragment-A关联上Activity之后将Activity强转为接口类型,然后在某个时刻Fragment中回调这个接口,然后再从Activity中调用Fragment-B中方法。

## 10、HOME键与BACK键

HOME键的执行顺序：onPause->onStop->onRestart->onStart->onResume

BACK键的顺序： onPause->onStop->onDestroy->onCreate->onStart->onResume

## 11、Android如何退出APP？？？

使用全局变量。对了，第一个想到的就是继承 Application，代码入下。（注意，值得商榷）

public class AgentApplication extends Application {

private List activities = new ArrayList();

public void addActivity(Activity activity) {

activities.add(activity);

}

@Override

public void onTerminate() {

super.onTerminate();

for (Activity activity : activities) {

activity.finish();

}

onDestroy();

System.exit(0);

}

}