《Android零基础入门课程》—— 涂涂IT学堂

第五章 Android应用开发核心知识点讲解

目标

- Android四大组件
- 数据存储
- 网络编程
- WebView
- Canvas
- 硬件与传感器

01 Android四大组件

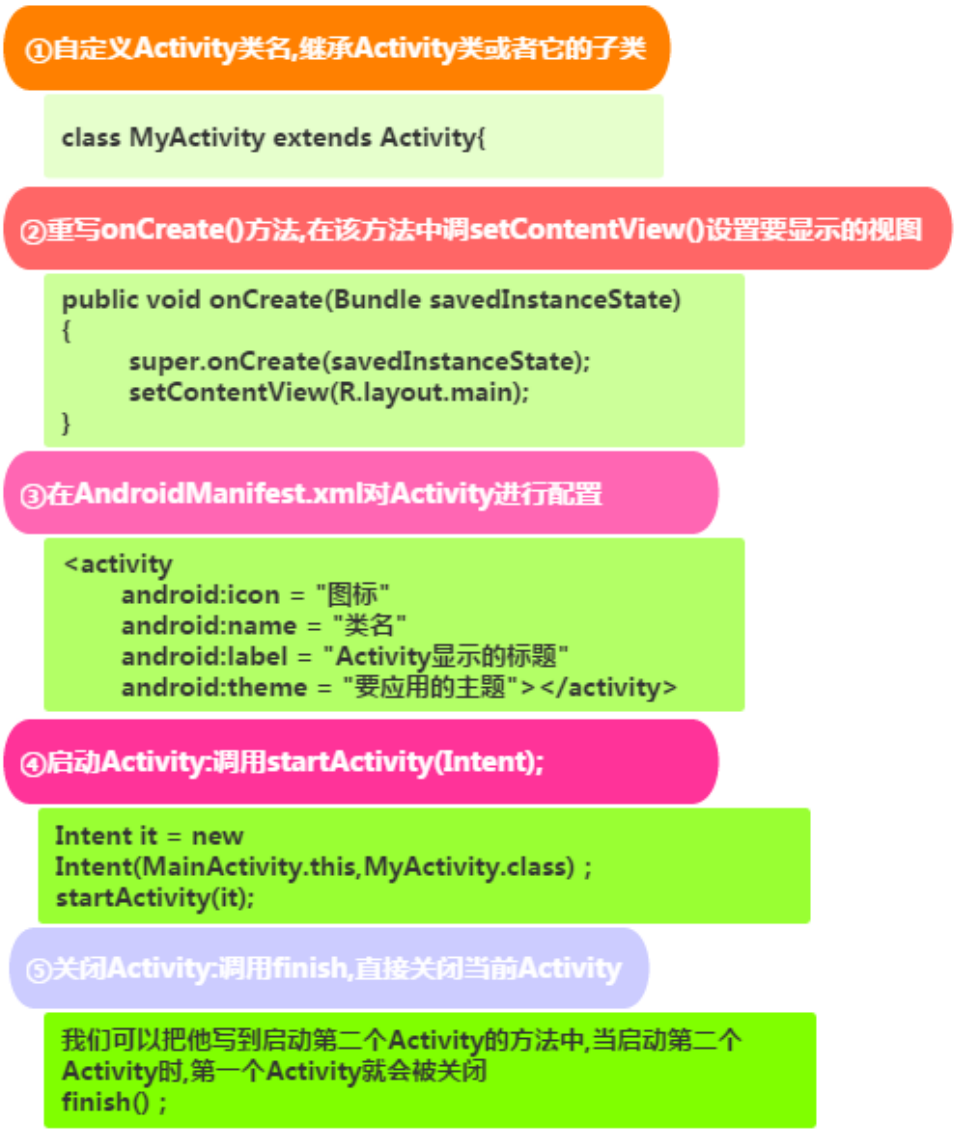
1.1 Activity(活动)

```
官方解释：Activity是一个应用程序的组件，他在屏幕上提供了一个区域，允许用户在上面做一些交互性的操作， 比如打电话，照相，发送邮件，或者显示一个地图！Activity可以理解成一个绘制用户界面的窗口， 而这个窗口可以填满整个屏幕，也可能比屏幕小或者浮动在其他窗口的上方！
总结：1. Activity用于显示用户界面，用户通过Activity交互完成相关操作 2. 一个App允许有多个Activity
```

```
继承Activity和AppCompatActivity区别
AppCompatActivity兼容了很多低版本的一些东西
AppcompaActivity相对于Activity的变化：主界面带有toolbar的标题栏；
```

- `Activity`创建流程

Activity的使用流程



- 启动一个Activity的几种方式

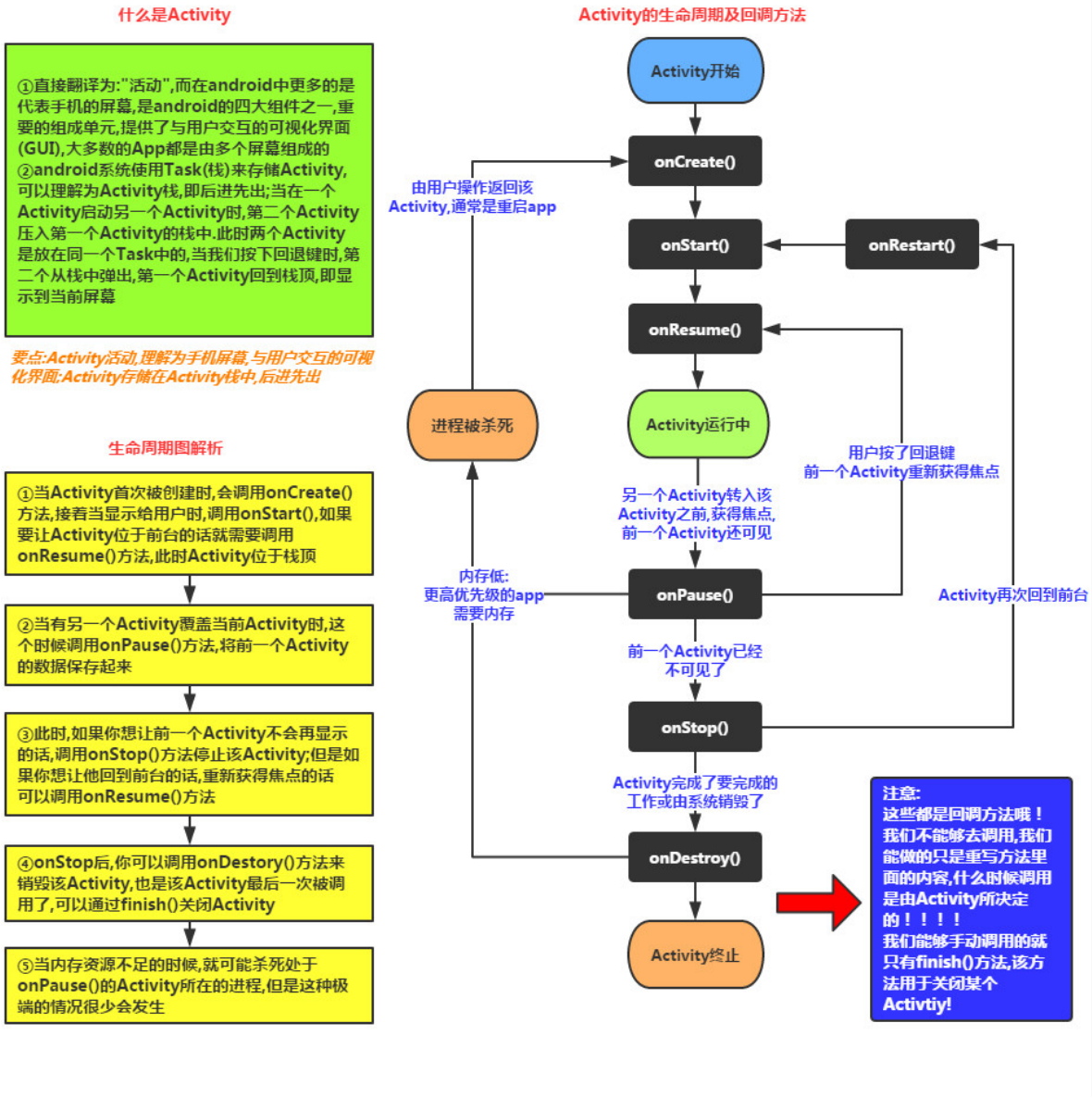
    1. 显示启动

```
①最常见的：
startActivity(new Intent(当前Act.this,要启动的Act.class));

②通过Intent的ComponentName：
ComponentName cn = new ComponentName("当前Act的全限定类名","启动Act的全限定类名") ;
Intent intent = new Intent() ;
intent.setComponent(cn) ;
startActivity(intent) ;

③初始化Intent时指定包名：
Intent intent = new Intent("android.intent.action.MAIN");
intent.setClassName("当前Act的全限定类名","启动Act的全限定类名");
startActivity(intent);
```

2. 隐世启动

```
通过Intent-filter的Action,Category或data来实现
```

- Activity 生命周期



- 组件间通信 Intent

```
Intent in = new Intent(FirstActivity.this, ThirdActivity.class);
//1.传单个数据
in.putExtra("test","TTIT");
in.putExtra("number",100);
//2.传多个数据
Bundle b = new Bundle();
b.putInt("number", 100);
b.putString("test", "TTIT");
in.putExtras(b);
startActivity(in);
```

```
1.FirstActivity启动ThirdActivity
```

```
    startActivityForResult(in, 1001);
2.FirstActivity接受ThirdActivity返回的数据
    @Override
        protected void onActivityResult(int requestCode, int
resultCode, @Nullable Intent data) {
            super.onActivityResult(requestCode, resultCode, data);
            Log.e("tag", "requestCode =" + requestCode);
            Log.e("tag", "resultCode =" + resultCode);
            Log.e("tag", "data =" + data.getStringExtra("back"));
        }
3.ThirdActivity设置返回的数据
    Intent backIn = new Intent();
    backIn.putExtra("back", "abcdef");
    setResult(1002, backIn);
```

- Back Stack（回退堆栈）

```
Java栈Stack概念：
后进先出(LIFO)，常用操作入栈(push)，出栈(pop)，处于最顶部的叫栈顶，最底部叫栈底
```

```
Activity 管理机制：
1.我们的APP一般都是由多个Activity构成的，而在Android中给我们提供了一个Task(任务)的概念， 就是将多个相关的Activity收集起来，然后进行Activity的跳转与返回；
2.当切换到新的Activity，那么该Activity会被压入栈中，成为栈顶！ 而当用户点击Back键，栈顶的Activity出栈，紧随其后的Activity来到栈顶！
3.Task是Activity的集合，是一个概念，实际使用的Back Stack来存储Activity，可以有多个Task，但是 同一时刻只有一个栈在最前面，其他的都在后台
```

```
1.FLAG_ACTIVITY_NEW_TASK
  默认启动标志，该标志控制创建一个新的Activity实例，首先会查找是否存在和被启动的Activity具有相同的亲和性的任务栈 如果有，则直接把这个栈整体移动到前台，并保持栈中旧activity的顺序不变，然后被启动的Activity会被压入栈，如果没有，则新建一个栈来存放被启动的activity
  Intent intent = new Intent(A.this, A.class);
  intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
  startActivity(intent);
2.FLAG_ACTIVITY_CLEAR_TOP
  如果已经启动了四个Activity：A，B，C和D。在D Activity里，我们要跳到B Activity，同时希望C finish掉,可以采用下面启动方式，这样启动B Activity，就会把D，C都finished掉
  Intent intent = new Intent(D.this, B.class);
  intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
  startActivity(intent);
3.FLAG_ACTIVITY_SINGLE_TOP
  从名字中不难看出该Flag相当于Activity加载模式中的singleTop模式，即原来Activity栈中有A、B、C、D这4个Activity实例，当在Activity D中再次启动Activity D时，Activity栈中依然还是A、B、C、D这4个Activity实例。
  Intent intent = new Intent(D.this, D.class);
  intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
  startActivity(intent);
```

Activity启动模式：

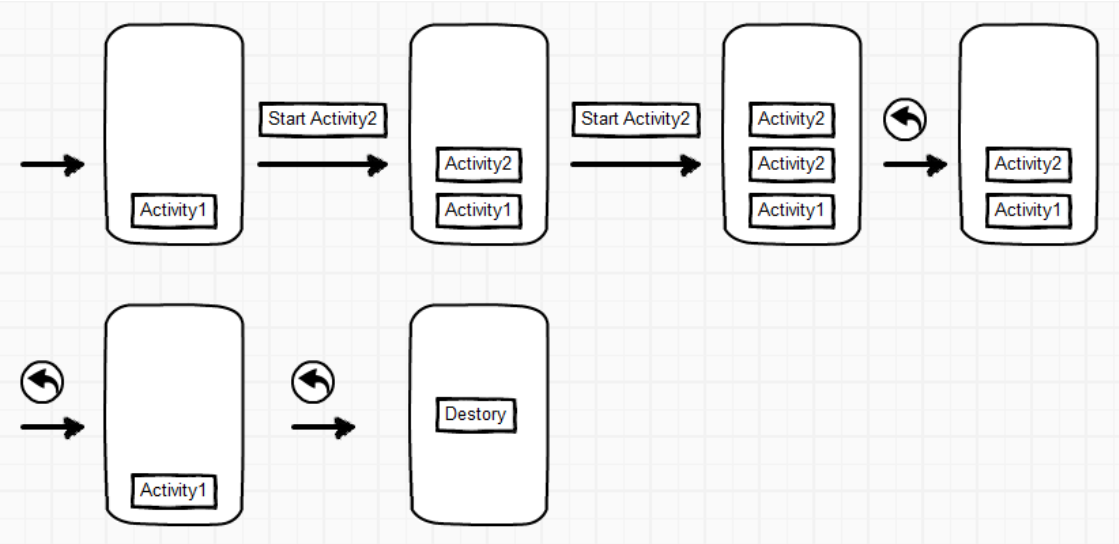
模式详解：
standard模式：
标准启动模式，也是activity的默认启动模式。在这种模式下启动的activity可以被多次实例化，即在同一个任务中可以存在多个activity的实例，每个实例都会处理一个Intent对象。如果Activity A的启动模式为standard，并且A已经启动，在A中再次启动Activity A，即调用startActivity（new Intent（this，A.class））， 会在A的上面再次启动一个A的实例，即当前的栈中的状态为A-->A。

singleTop模式：
如果一个以singleTop模式启动的Activity的实例已经存在于任务栈的栈顶， 那么再启动这个Activity时，不会创建新的实例，而是重用位于栈顶的那个实例， 并且会调用该实例的onNewIntent()方法将Intent对象传递到这个实例中。 举例来说，如果A的启动模式为singleTop，并且A的一个实例已经存在于栈顶中， 那么再调用startActivity（new Intent（this，A.class））启动A时， 不会再次创建A的实例，而是重用原来的实例，并且调用原来实例的onNewIntent()方法。 这时任务栈中还是这有一个A的实例。如果以singleTop模式启动的activity的一个实例 已经存在与任务栈中，但是不在栈顶，那么它的行为和standard模式相同，也会创建多个实例。

singleTask模式：
只允许在系统中有一个Activity实例。如果系统中已经有了一个实例， 持有这个实例的任务将移动到顶部，同时intent将被通过onNewIntent()发送。 如果没有，则会创建一个新的Activity并置放在合适的任务中。

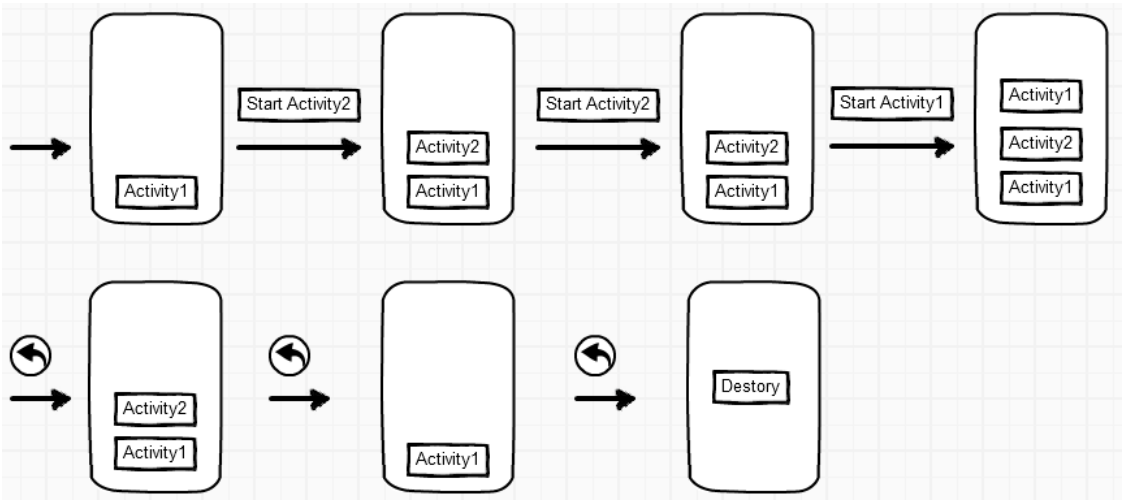
singleInstance模式：
保证系统无论从哪个Task启动Activity都只会创建一个Activity实例,并将它加入新的Task栈顶 也就是说被该实例启动的其他activity会自动运行于另一个Task中。 当再次启动该activity的实例时，会重用已存在的任务和实例。并且会调用这个实例 的onNewIntent()方法，将Intent实例传递到该实例中。和singleTask相同， 同一时刻在系统中只会存在一个这样的Activity实例。

standard模式:



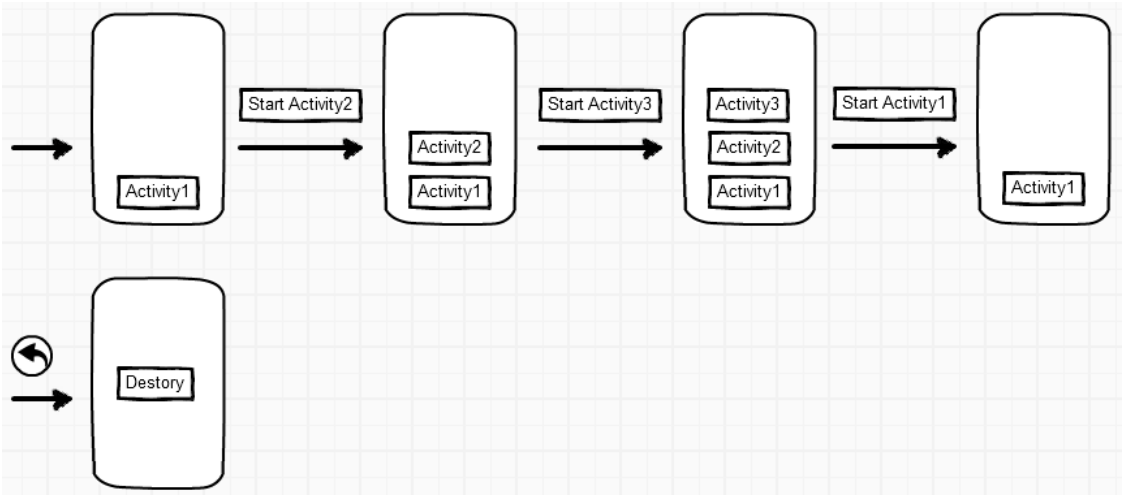
singleTop模式：

在该模式下，如果栈顶Activity为我们要新建的Activity（目标Activity），那么就不会重复创建新的Activity。



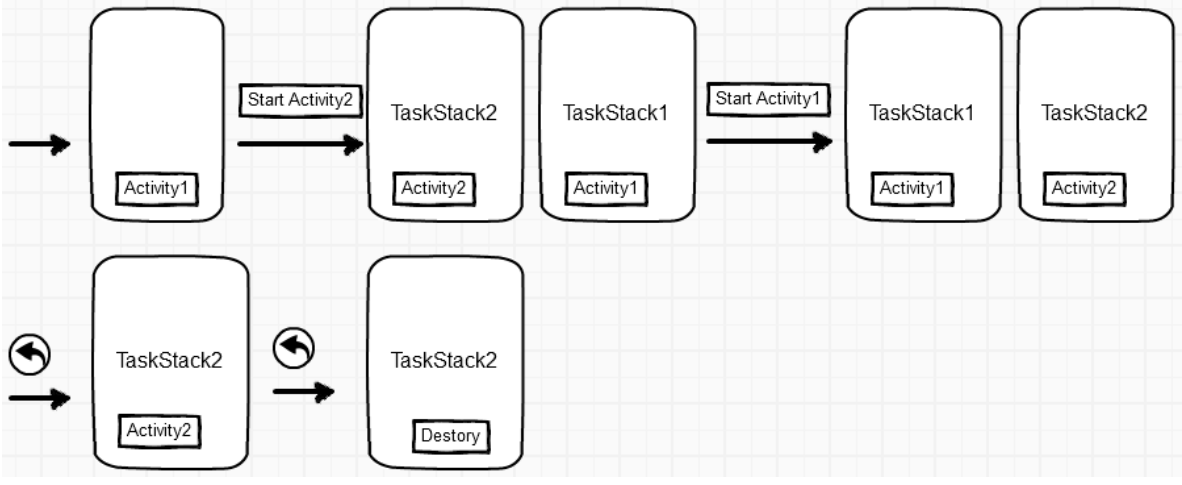
singleTask模式：

1 与singleTop模式相似，只不过singleTop模式是只是针对栈顶的元素，而singleTask模式下，如果task栈内存在目标Activity实例，则：
2 将task内的对应Activity实例之上的所有Activity弹出栈。
3 将对应Activity置于栈顶，获得焦点。



singleInstance（全局唯一）模式：

1 是我们最后的一种启动模式，也是我们最恶心的一种模式：在该模式下，我们会为目标Activity分配一个新的affinity，并创建一个新的Task栈，将目标Activity放入新的Task，并让目标Activity获得焦点。新的Task有且只有这一个Activity实例。　　如果已经创建过目标Activity实例，则不会创建新的Task，而是将以前创建过的Activity唤醒（对应Task设为Foreground状态）

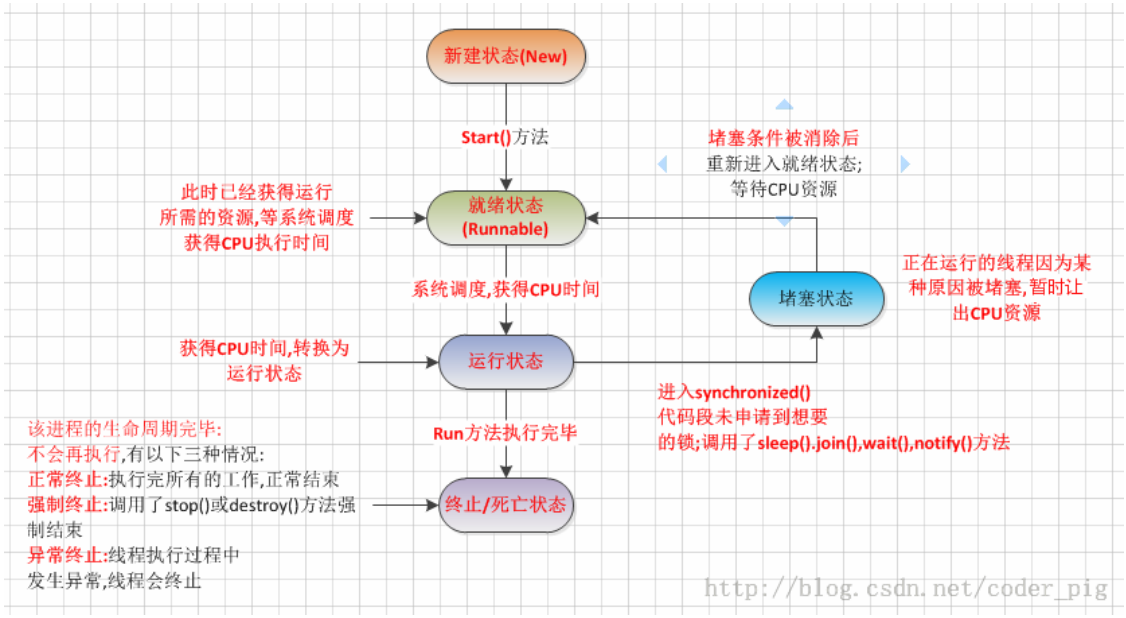


1.2 Service(服务)

- 线程的相关概念

  - **程序**：为了完成特定任务，用某种语言编写的一组指令集合(一组**静态代码**)
  - **进程**：**运行中的程序**，系统调度与资源分配的一个**独立单位**，操作系统会 为每个进程分配一段内存空间！程序的依次动态执行，经历代码的加载，执行， 执行完毕的完整过程！
  - **线程**：比进程更小的执行单元，每个进程可能有多条线程，**线程**需要放在一个 **进程**中才能执行，**线程由程序**负责管理，而**进程则由系统**进行调度！
  - **多线程的理解**：**并行**执行多个条指令，将**CPU时间片**按照调度算法分配给各个 线程，实际上是**分时**执行的，只是这个切换的时间很短，用户感觉到"同时"而已！

- 线程的生命周期



- 创建线程的三种方式

  1. 继承Thread类

```
1  public class MyThread extends Thread{
2
3       @Override
4      public void run() {
5          // TODO Auto-generated method stub
6          //super.run();
7           doSomething();
8      }
9
10     private void doSomething() {
11         // TODO Auto-generated method stub
12         System.out.println("我是一个线程中的方法");
13     }
14 }
15 ================================================================
   ===============
16 public class NewThread {
17     public static void main(String[] args) {
18         MyThread myThread=new MyThread();
19         myThread.start();//开启一个线程方法
20         //以下的方法可与上边的线程并发执行
21         doSomething();
```

```
    }

    private static void doSomething() {
        // TODO Auto-generated method stub
    }
}
```

2. 实现Runnable接口

```
public class RunnableThread implements Runnable{

    @Override
    public void run() {
        // TODO Auto-generated method stub
        doSomeThing();
    }

    private void doSomeThing() {
        // TODO Auto-generated method stub
        System.out.println("我是一个线程方法");
    }
}
================================================================
===============
public class NewThread {
    public static void main(String[] args) {
        Runnable runnable=new RunnableThread();
        Thread thread=new Thread(runnable);
        thread.start();//开启一个线程方法
            //以下的方法可与上边的线程并发执行
        doSomething();
    }

    private static void doSomething() {
        // TODO Auto-generated method stub
    }
}
```

3. 实现Callable接口和Future创建线程

```
public class CallableThread implements Callable<String>{

    @Override
    public String call() throws Exception {
        // TODO Auto-generated method stub
        doSomeThing();
        return "需要返回的值";
    }

    private void doSomeThing() {
        // TODO Auto-generated method stub
        System.out.println("我是线程中的方法");
    }
```

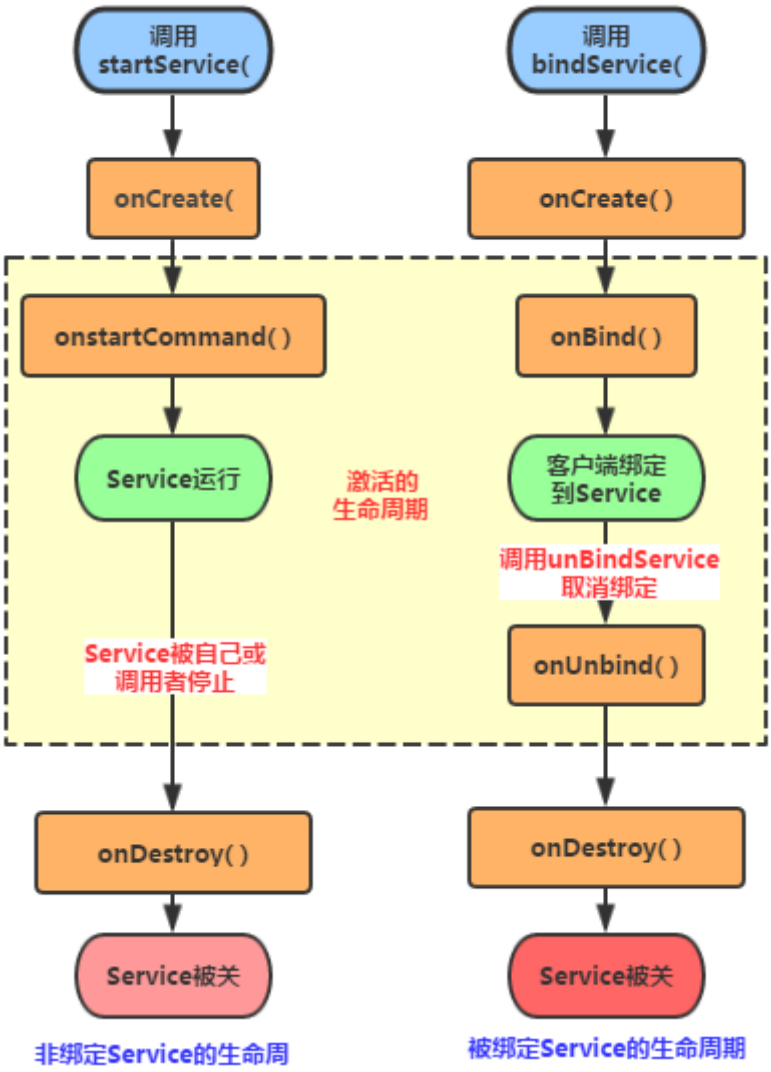
```
}
================================================================================
public class NewThread {
    public static void main(String[] args) {
        Callable<String> callable=new CallableThread();
        FutureTask<String> futureTask=new FutureTask<String>(callable);
        Thread thread=new Thread(futureTask);
        thread.start();//开启一个线程方法
            //以下的方法可与上边的线程并发执行
        doSomething();
        try {
            futureTask.get();//获取线程返回值
        } catch (InterruptedException | ExecutionException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private static void doSomething() {
        // TODO Auto-generated method stub
    }
}
```

4. Service与Thread线程的区别

> 其实他们两者并没有太大的关系，不过有很多朋友经常把这两个混淆了！ Thread是线程，程序执行的最小单元，分配CPU的基本单位！ 而Service则是Android提供一个允许长时间留驻后台的一个组件，最常见的 用法就是做轮询操作！或者想在后台做一些事情，比如后台下载更新！ 记得别把这两个概念混淆！

- Service的生命周期

Service的生命周期图




```
生命周期函数解析：
1）onCreate()：当Service第一次被创建后立即回调该方法，该方法在整个生命周期 中只会调用一次！
2）onDestory()：当Service被关闭时会回调该方法，该方法只会回调一次！
3）onStartCommand(intent,flag,startId)：早期版本是onStart(intent,startId)， 当客户端调用startService(Intent)方法时会回调，可多次调用StartService方法， 但不会再创建新的Service对象，而是继续复用前面产生的Service对象，但会继续回调 onStartCommand()方法！
IBinder onOnbind(intent)：该方法是Service都必须实现的方法，该方法会返回一个 IBinder 对象，app通过该对象与Service组件进行通信！
4）onUnbind(intent)：当该Service上绑定的所有客户端都断开时会回调该方法！
```

- service启动方式：

```
1）StartService()启动Service
2）BindService()启动Service
PS:还有一种，就是启动Service后，绑定Service！
```

- StartService启动Service

```
①首次启动会创建一个Service实例,依次调用onCreate()和onStartCommand()方法,此时Service 进入运行状态,如果再次调用StartService启动Service,将不会再创建新的Service对象, 系统会直接复用前面创建的Service对象,调用它的onStartCommand()方法!
②但这样的Service与它的调用者无必然的联系,就是说当调用者结束了自己的生命周期, 但是只要不调用stopService,那么Service还是会继续运行的!
③无论启动了多少次Service,只需调用一次StopService即可停掉Service
```

- BindService启动Service

```
①当首次使用bindService绑定一个Service时,系统会实例化一个Service实例,并调用其onCreate()和onBind()方法,然后调用者就可以通过IBinder和Service进行交互了,此后如果再次使用bindService绑定Service,系统不会创建新的Sevice实例,也不会再调用onBind()方法,只会直接把IBinder对象传递给其他后来增加的客户端!
②如果我们解除与服务的绑定,只需调用unbindService(),此时onUnbind和onDestory方法将会被调用!这是一个客户端的情况,假如是多个客户端绑定同一个Service的话,情况如下 当一个客户完成和service之间的互动后, 它调用 unbindService() 方法来解除绑定。当所有的客户端都和service解除绑定后, 系统会销毁service。(除非service也被startService()方法开启)
③另外,和上面那张情况不同,bindService模式下的Service是与调用者相互关联的,可以理解为"一条绳子上的蚂蚱",要死一起死,在bindService后,一旦调用者销毁,那么Service也立即终止!
通过BindService调用Service时调用的Context的bindService的解析
bindService(Intent Service,ServiceConnection conn,int flags)
service:通过该intent指定要启动的Service
conn:ServiceConnection对象,用户监听访问者与Service间的连接情况, 连接成功回调该对象中的onServiceConnected(ComponentName,IBinder)方法; 如果Service所在的宿主由于异常终止或者其他原因终止,导致Service与访问者间断开 连接时调用onServiceDisconnected(CompanentName)方法,主动通过unBindService() 方法断开并不会调用上述方法!
flags:指定绑定时是否自动创建Service(如果Service还未创建), 参数可以是0(不自动创建),BIND_AUTO_CREATE(自动创建)
```

- StartService启动Service后bindService绑定

```
如果Service已经由某个客户端通过StartService()启动,接下来由其他客户端 再调用bindService()绑定到该Service后调用unbindService()解除绑定最后在 调用bindService()绑定到Service的话,此时所触发的生命周期方法如下:
onCreate( )->onStartCommand( )->onBind( )->onUnbind( )->onRebind( )
PS:前提是:onUnbind()方法返回true!!! 这里或许部分读者有疑惑了,调用了unbindService后Service不是应该调用 onDistory()方法么!其实这是因为这个Service是由我们的StartService来启动的 ,所以你调用onUnbind()方法取消绑定,Service也是不会终止的!
得出的结论: 假如我们使用bindService来绑定一个启动的Service,注意是已经启动的Service!!! 系统只是将Service的内部IBinder对象传递给Activity,并不会将Service的生命周期 与Activity绑定,因此调用unBindService( )方法取消绑定时,Service也不会被销毁!
```

1.3 BroadcastReceiver 广播接收器

- 前言

```
为了方便Android系统各个应用程序及程序内部进行通信, Android系统引入了一套广播机制。各个应用程序可以对感兴趣的广播进行注册, 当系统或者其他程序发出这条广播的时候, 对发出的广播进行注册的程序便能够收到这条广播。为此, Android系统中有一套完整的API, 允许程序只有的发送和接受广播。
```

- 在Android系统中，主要有两种基本的广播类型：
  - 标准广播（Normal Broadcasts）

```
是一种完全异步执行的广播，在广播发出之后，所有的广播接收器会在同一时间接收到这条广播，广播无法被截断.
发送方式：
Intent intent=new Intent("com.example.dimple.BROADCAST_TEST");
sendBroadcast(intent);
```

  - 有序广播（Ordered Broadcasts）

```
是一种同步执行的广播，在广播发出之后，优先级高的广播接收器可以优先接收到这条广播，并可以在优先级较低的广播接收器之前截断停止发送这条广播。
发送方式：
Intent intent=new Intent("com.example.dimple.BROADCAST_TEST");
sendOrderBroadcast(intent, null);//第二个参数是与权限相关的字符串。
```

- 注册广播

```
在Android的广播接收机制中，如果需要接收广播，就需要创建广播接收器。而创建广播接收器的方法就是新建一个类（可以是单独新建类，也可以是内部类（public）） 继承自BroadcastReceiver

class myBroadcastReceiver extends BroadcastReceiver{

        @Override
        public void onReceive(Context context, Intent intent) {
            //不要在广播里添加过多逻辑或者进行任何耗时操作,因为在广播中是不允许开辟线程的, 当onReceiver( )方法运行较长时间(超过10秒)还没有结束的话,那么程序会报错(ANR), 广播更多的时候扮演的是一个打开其他组件的角色,比如启动Service,Notification提示, Activity等!
        }
}
```

- 动态注册和静态注册的区别

```
动态注册的广播接收器可以自由的控制注册和取消，有很大的灵活性。但是只能在程序启动之后才能收到广播，此外，不知道你注意到了没，广播接收器的注销是在onDestroy()方法中的。所以广播接收器的生命周期是和当前活动的生命周期一样。
静态注册的广播不受程序是否启动的约束，当应用程序关闭之后，还是可以接收到广播。
```

- 两种方式注册广播：
  - 动态注册

```
所谓动态注册是指在代码中注册。步骤如下 ：
- 实例化自定义的广播接收器。
- 创建IntentFilter实例。
- 调用IntentFilter实例的addAction()方法添加监听的广播类型。
- 最后调用Context的registerReceiver(BroadcastReceiver,IntentFilter)动态的注册广播。
PS:这里提醒一点，如果需要接收系统的广播（比如电量变化，网络变化等等），别忘记在AndroidManifest配置文件中加上权限。

另外，动态注册的广播在活动结束的时候需要取消注册：
 @Override
 protected void onDestroy() {
     super.onDestroy();
     unregisterReceiver(myBroadcastReceiver);
 }
```

- 静态注册

```
<receiver
android:name="com.ttit.core.broadcastreceiver.MyBRReceiver2">
    <intent-filter>
        <action
android:name="com.example.broadcasttest.MY_BROADCAST" />
    </intent-filter>
</receiver>
```

1.4 ContentProvider(内容提供者)

- ContentProvider应用场景：
  我们想在自己的应用中访问别的应用，或者说一些ContentProvider暴露给我们的一些数据， 比如手机联系人，短信等！我们想对这些数据进行读取或者修改，这就需要用到ContentProvider了！

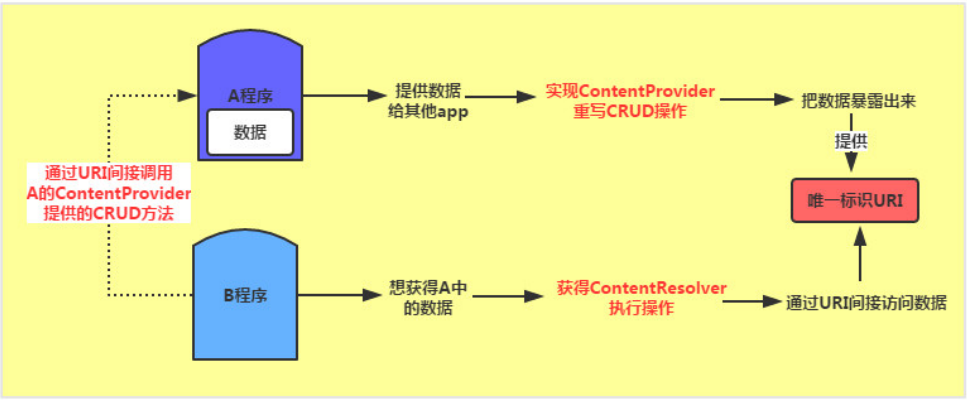
  我们自己的应用，想把自己的一些数据暴露出来，给其他的应用进行读取或操作，我们也可以用到ContentProvider，另外我们可以选择要暴露的数据，就避免了我们隐私数据的的泄露！
- ContentProvider概念讲解

ContentProvider(内容提供者)

ContentProvider的概述:

当我们想允许自己的应用的数据允许别的应用进行读取操作,我们可以让我们的App实现ContentProvider类,同时注册一个Uri,然后其他应用只要使用ContentResolver根据Uri就可以操作我们的app中的数据了！而数据不一定是数据库,也可能是文件,xml或者其他,但是SharedPreference使用基于数据库模型的简单表格来提供其中的数据！

ContentProvider的执行原理



URI简介:

专业名词叫做:通用资源标识符,而你也可以类比为网页的域名,我们暂且就把他叫做资源定位符吧,就是定位资源所在路径的
而在本节ContentProvider中,Uri灰常重要,我们分析一个简单的例子吧:
content://com.jay.example.providers.myprovider/word/2

分析:

content:协议头,这个是规定的,就像http,ftp等一样,规定的,而ContentProvider规定的是content开头的
接着是provider所在的全限定类名
word:代表资源部分,如果想访问word所有资源,后面的2就不用写了,直接写word
2:访问的是word资源中id为2的记录

附加

当然,上面也说过数据不仅仅来自于数据库,有时也来源于文件,xml或者网络等其他存储方式,
但是依旧可以使用上面这种URI定义方式:
比如：当表示的xml文件时:~/word/detail表示word节点下的detail结点
另外:URI还提供一个parse()方法将字符串转换为URI
eg:Uri uri = Uri.parese("Content://~");

- ContentProvider的URI

http://blog.csdn.net/

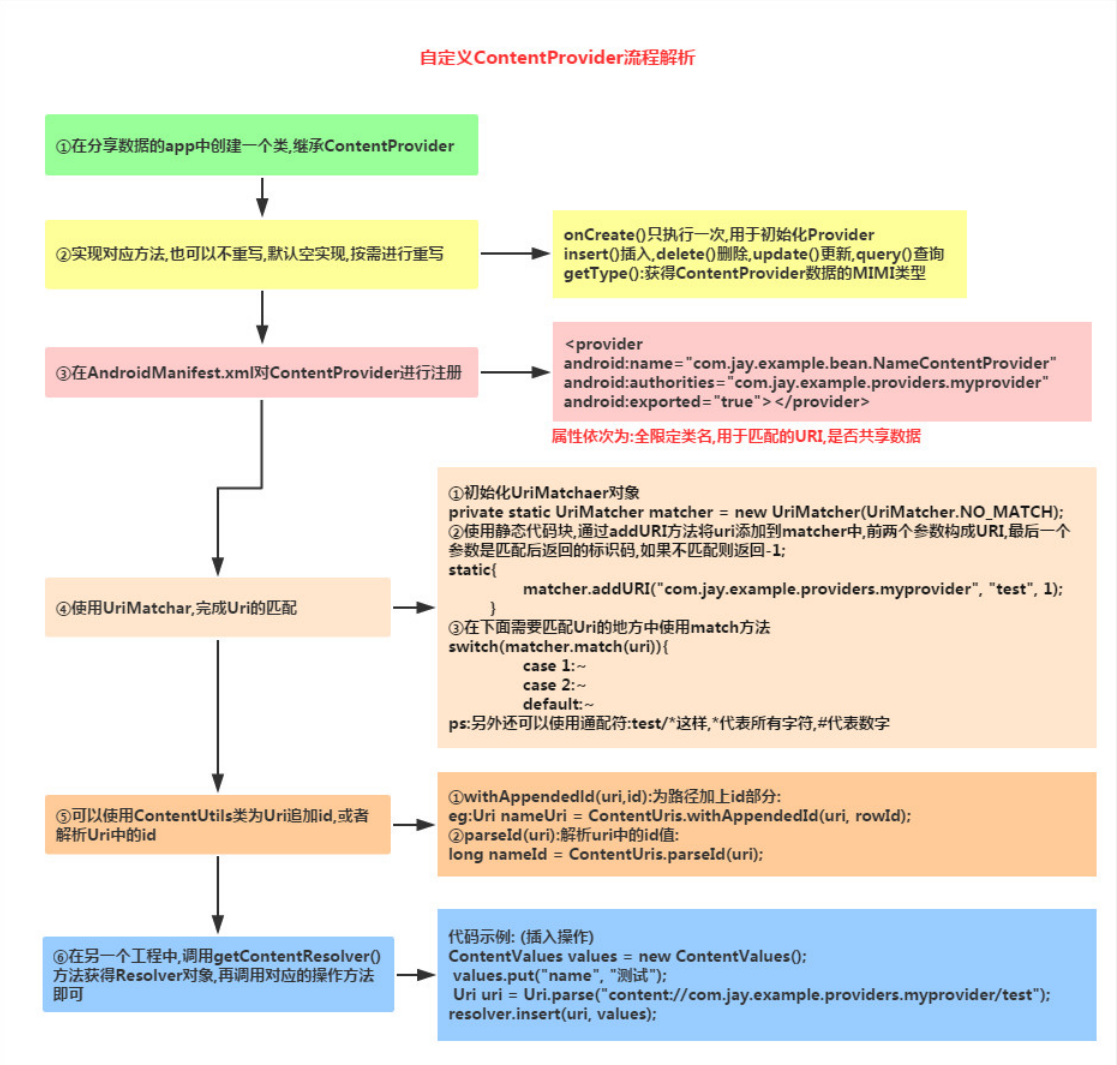
主要分三个部分：scheme，authority and path。scheme表示上图中的content://，authority表示B部分，path表示C和D部分。
A部分：表示是一个Android内容URI，说明由ContentProvider控制数据，该部分是固定形式，不可更改的。
B部分：是URI的授权部分，是唯一标识符，用来定位ContentProvider。格式一般是自定义ContentProvider类的完全限定名称，注册时需要用到，如：com.example.transportationprovider
C部分和D部分：是每个ContentProvider内部的路径部分，C和D部分称为路径片段，C部分指向一个对象集合，一般用表的名字，如：/trains表示一个笔记集合；D部分指向特定的记录，如：/trains/122表示id为122的单条记录，如果没有指定D部分，则返回全部记录。

- 使用系统提供的ContentProvider

打开模拟器的file exploer/data/data/com.android.providers.contacts/databases/contact2.db 导出后使用SQLite图形工具查看，三个核心的表:raw_contact表，data表，mimetypes表

- 自定义ContentProvider

我们自己的应用，想把自己的一些数据暴露出来，给其他的应用进行读取或操作，我们也可以用 到ContentProvider，另外我们可以选择要暴露的数据，就避免了我们隐私数据的的泄露！



1.5 Intent (意图)

- 四大组件间的 枢纽——Intent(意图)，Android通信的桥梁

```
startActivity(Intent)/startActivityForResult(Intent)：来启动一个Activity
startService(Intent)/bindService(Intent)：来启动一个Service
sendBroadcast：发送广播到指定BroadcastReceiver
另外我们在注册四大组件时，写得很多的Intent-Filter
```

- 显式Intent与隐式Intent

  显式Intent：通过组件名指定启动的目标组件,比如startActivity(new Intent(A.this,B.class)); 每次**启动的组件只有一个**

  隐式Intent:不指定组件名,而指定Intent的Action,Data,或Category,当我们启动组件时, 会去匹配AndroidManifest.xml相关组件的Intent-filter,逐一匹配出满足属性的组件,**当不止一个满足时, 会弹出一个让我们选择启动哪个的对话框**

- Intent 中包含的主要信息如下：

  1. ComponentName: 组件名称
     指定Intent的目标组件的类名称。组件名称是可选的，如果填写，Intent对象会发送给指定组件名称的组件，否则也可以通过其他Intent信息定位到适合的组件。组件名称是个ComponentName类型的对象

```
  Intent intent = new Intent();
  ComponentName cn = new ComponentName(IntentActivity.this,
OtherActivity.class);
  in.setComponent(cn);
  startActivity(intent);
```

  2. action : 该activity可以执行的动作
     该标识用来说明这个activity可以执行哪些动作，所以当隐式intent传递过来action时，如果跟这里所列出的任意一个匹配的话，就说明这个activity是可以完成这个intent的意图的，可以将它激活！

  常用的Action如下所示：

> ACTION_CALL activity 启动一个电话.
>
> ACTION_EDIT activity 显示用户编辑的数据.
>
> ACTION_MAIN activity 作为Task中第一个Activity启动
>
> ACTION_SYNC activity 同步手机与数据服务器上的数据.
>
> ACTION_BATTERY_LOW broadcast receiver 电池电量过低警告.
>
> ACTION_HEADSET_PLUG broadcast receiver 插拔耳机警告
>
> ACTION_SCREEN_ON broadcast receiver 屏幕变亮警告.
>
> ACTION_TIMEZONE_CHANGED broadcast receiver 改变时区警告.

  两条原则：

```
 一条<intent-filter>元素至少应该包含一个<action>，否则任何Intent请求都不能和该<intent-filter>匹配。

 如果Intent请求的Action和<intent-filter>中个任意一条<action>匹配，那么该Intent就可以激活该activity(前提是除了action的其它项也要通过)。
```

  两条注意：

> 如果Intent请求或<intent-filter>中没有说明具体的Action类型，那么会出现下面两种情况。如果<intent-filter>中没有包含任何Action类型，那么无论什么Intent请求都无法和这条<intent-filter>匹配。 反之，如果Intent请求中没有设定Action类型，那么只要<intent-filter>中包含有Action类型，这个Intent请求就将顺利地通过<intent-filter>的行为测试。

3. data: 执行时要操作的数据

在目标标签中包含了以下几种子元素，他们定义了url的匹配规则：

android:scheme 匹配url中的前缀，除了“http”、“https”、“tel”...之外，我们可以定义自己的前缀

android:host 匹配url中的主机名部分，如“google.com”，如果定义为“*”则表示任意主机名

android:port 匹配url中的端口

android:path 匹配url中的路径

```
<activity android:name=".TargetActivity">
    <intent-filter>
        <action android:name="com.scott.intent.action.TARGET"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:scheme="scott"
              android:host="com.scott.intent.data"
              android:port="7788"
              android:path="/target"/>
    </intent-filter>
</activity>
```

注意：

这个标识比较特殊，它定义了执行此Activity时所需要的数据，也就是说，这些数据是必须的！！！！！所有如果其它条件都足以激活该Activity，但intent却没有传进来指定类型的Data时，就不能激活该activity！！！！

4. Category: 指定当前动作（Action）被执行的环境

即这个activity在哪个环境中才能被激活。不属于这个环境的，不能被激活。

常用的Category属性如下所示：

> CATEGORY_DEFAULT：Android系统中默认的执行方式，按照普通Activity的执行方式执行。表示所有intent都可以激活它
>
> CATEGORY_HOME：设置该组件为Home Activity。
>
> CATEGORY_PREFERENCE：设置该组件为Preference。
>
> CATEGORY_LAUNCHER：设置该组件为在当前应用程序启动器中优先级最高的Activity，通常为入口ACTION_MAIN配合使用。
>
> CATEGORY_BROWSABLE：设置该组件可以使用浏览器启动。表示该activity只能用来浏览网页。
>
> CATEGORY_GADGET：设置该组件可以内嵌到另外的Activity中。

注意：如果该activity想要通过隐式intent方式激活，那么不能没有任何category设置，至少包含一个android.intent.category.DEFAULT

5. Extra

Extras属性主要用于传递目标组件所需要的额外的数据。通过putExtras()方法设置

常用值如下所示：

> EXTRA_BCC：存放邮件密送人地址的字符串数组。

> EXTRA_CC：存放邮件抄送人地址的字符串数组。
>
> EXTRA_EMAIL：存放邮件地址的字符串数组。
>
> EXTRA_SUBJECT：存放邮件主题字符串。
>
> EXTRA_TEXT：存放邮件内容。
>
> EXTRA_KEY_EVENT：以KeyEvent对象方式存放触发Intent的按键。
>
> EXTRA_PHONE_NUMBER：存放调用ACTION_CALL时的电话号码

6. IntentFlag
   在 Intent 类中定义的、充当 Intent 元数据的标志。标志可以指示 Android 系统如何启动 Activity（例如，Activity 应属于哪个任务），以及启动之后如何处理（例如，它是否属于最近的 Activity 列表）。具体可以查看我之前写的一篇文章。IntentFlag大全及使用总结

显式 Intent 示例：

```
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

隐式 Intent 示例：

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

- Intent Filter匹配规则

  Intent解析机制主要是通过查找已注册在AndroidManifest.xml中的所有IntentFilter及其中定义的Intent，最终找到匹配的Intent。一个组件可以声明多个Intent Filter，只需要匹配任意一个即可启动该组件。 一个Intent Filter中的action、type、category可以有多个，所有的action、type、category分别构成不同类别，同一类别信息共同约束当前类别的匹配过程。只有一个Intent同时匹配一个Intent Filter的action、type、category这三个类别才算完全匹配，只有完全匹配才能启动Activity。
  比如下面定义了两个intent-filter，只要有一个Intent Filter的action、type、category完全匹配即可：

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="http" android:mimeType="video/*" />
</intent-filter>

<intent-filter>
    <action android:name="com.study.jankin.test"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="demoapp"  />
</intent-filter>
```

  1. action的匹配规则

一个Intent Filter中可声明多个action，Intent中的action与其中的任一个action在字符串形式上完全相同（注意，**区分大小写**，大小写不同但字符串内容相同也会造成匹配失败），action方面就匹配成功。

注意，隐式Intent必须指定action。

比如我们在Manifest文件中为Activity定义了如下Intent Filter：

```
<intent-filter>
    <action android:name="com.study.jankin.test"/>
</intent-filter>
```

在程序中我们就可以通过下面的代码启动该Activity

```
Intent intent = new Intent("com.study.jankin.test");
startActivity(intent);
```

2. category的匹配规则

category也是一个字符串，但是它与action的过滤规则不同，它要求Intent中个如果含有category，那么所有的category都必须和过滤规则中的其中一个category相同。也就是说，Intent中如果出现了category，不管有几个category，对于每个category来说，它必须是过滤规则中的定义了的category。当然，Intent中也可以没有category（若Intent中未指定category，系统会自动为它带上“android.intent.category.DEFAULT”），如果没有，仍然可以匹配成功。我们可以通过addCategory方法为Intent添加category。
如常用的 可以让组件通过浏览器启动。

3. data的匹配规则

同action类似，如果过滤规则中定义了data, 那么Intent中必须也要定义可匹配的data，只要Intent的data与Intent Filter中的任一个data声明完全相同，data方面就完全匹配成功。
data由两部分组成：mimeType和Uri。

> **MineType**指的是媒体类型：例如imgage/jpeg，auto/mpeg4和viedo/*等，可以表示图片、文本、视频等不同的媒体格式
>
> **Uri** 可配置更多信息，类似于url。Uri的默认scheme是content或file。

URI的结构:

```
<scheme>://<host>:<port>/[<path>|<pathPrefix>|<pathPattern>]
content://com.axe.mg:100/fold/subfolder/etc
http://www.axe.com:500/profile/info
```

Uri的属性说明

> Scheme：URI的模式，比如http、file、content。如果URI中没有指定Scheme，那么整个URI无效。默认值为content 和 file。
> Host：URI的主机名。比如www.baidu.com，如果host未指定，那么整个Uri中的其他参数无效，这也意味着Uri是无效的。
> Port：URI端口，当URI指定了scheme 和 host 参数时port参数才有意义。
> path：用来匹配完整的路径，如：http://example.com/blog/abc.html，这里将 path 设置为 /blog/abc.html 才能够进行匹配；
> pathPrefix： 用来匹配路径的开头部分，拿上面的 Uri 来说，这里将 pathPrefix 设置为 /blog 就能进行匹配了；
> pathPattern： 用表达式来匹配整个路径。这里需要说下匹配符号与转义。

- 常用系统Intent合集

```
//=================================================================
//1.拨打电话
// 给移动客服10086拨打电话
Uri uri = Uri.parse("tel:10086");
Intent intent = new Intent(Intent.ACTION_DIAL, uri);
startActivity(intent);

//=================================================================

//2.发送短信
// 给10086发送内容为“Hello”的短信
Uri uri = Uri.parse("smsto:10086");
Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
intent.putExtra("sms_body", "Hello");
startActivity(intent);

//3.发送彩信（相当于发送带附件的短信）
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra("sms_body", "Hello");
Uri uri = Uri.parse("content://media/external/images/media/23");
intent.putExtra(Intent.EXTRA_STREAM, uri);
intent.setType("image/png");
startActivity(intent);

//=================================================================

//4.打开浏览器:
// 打开百度主页
Uri uri = Uri.parse("http://www.baidu.com");
Intent intent  = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);

//=================================================================

//5.发送电子邮件:(阉割了Google服务的没戏!!!!)
// 给someone@domain.com发邮件
Uri uri = Uri.parse("mailto:someone@domain.com");
Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
startActivity(intent);
// 给someone@domain.com发邮件发送内容为“Hello”的邮件
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, "someone@domain.com");
intent.putExtra(Intent.EXTRA_SUBJECT, "Subject");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
intent.setType("text/plain");
startActivity(intent);
// 给多人发邮件
Intent intent=new Intent(Intent.ACTION_SEND);
String[] tos = {"1@abc.com", "2@abc.com"}; // 收件人
String[] ccs = {"3@abc.com", "4@abc.com"}; // 抄送
String[] bccs = {"5@abc.com", "6@abc.com"}; // 密送
intent.putExtra(Intent.EXTRA_EMAIL, tos);
intent.putExtra(Intent.EXTRA_CC, ccs);
intent.putExtra(Intent.EXTRA_BCC, bccs);
intent.putExtra(Intent.EXTRA_SUBJECT, "Subject");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
intent.setType("message/rfc822");
```

```
startActivity(intent);

//=============================================================

//6.显示地图:
// 打开Google地图中国北京位置（北纬39.9，东经116.3）
Uri uri = Uri.parse("geo:39.9,116.3");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);

//=============================================================

//7.路径规划
// 路径规划：从北京某地（北纬39.9，东经116.3）到上海某地（北纬31.2，东经121.4）
Uri uri = Uri.parse("http://maps.google.com/maps?f=d&saddr=39.9
116.3&daddr=31.2 121.4");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);

//=============================================================

//8.多媒体播放:
Intent intent = new Intent(Intent.ACTION_VIEW);
Uri uri = Uri.parse("file:///sdcard/foo.mp3");
intent.setDataAndType(uri, "audio/mp3");
startActivity(intent);

//获取SD卡下所有音频文件,然后播放第一首=-=
Uri uri =
Uri.withAppendedPath(MediaStore.Audio.Media.INTERNAL_CONTENT_URI,
"1");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);

//=============================================================

//9.打开摄像头拍照:
// 打开拍照程序
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, 0);
// 取出照片数据
Bundle extras = intent.getExtras();
Bitmap bitmap = (Bitmap) extras.get("data");

//另一种:
//调用系统相机应用程序，并存储拍下来的照片
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
time = Calendar.getInstance().getTimeInMillis();
intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(new
File(Environment
.getExternalStorageDirectory().getAbsolutePath()+"/tucue", time +
".jpg")));
startActivityForResult(intent, ACTIVITY_GET_CAMERA_IMAGE);

//=============================================================

//10.获取并剪切图片
// 获取并剪切图片
```

```
Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
intent.setType("image/*");
intent.putExtra("crop", "true"); // 开启剪切
intent.putExtra("aspectX", 1); // 剪切的宽高比为1：2
intent.putExtra("aspectY", 2);
intent.putExtra("outputX", 20); // 保存图片的宽和高
intent.putExtra("outputY", 40);
intent.putExtra("output", Uri.fromFile(new File("/mnt/sdcard/temp")));
// 保存路径
intent.putExtra("outputFormat", "JPEG");// 返回格式
startActivityForResult(intent, 0);
// 剪切特定图片
Intent intent = new Intent("com.android.camera.action.CROP");
intent.setClassName("com.android.camera",
"com.android.camera.CropImage");
intent.setData(Uri.fromFile(new File("/mnt/sdcard/temp")));
intent.putExtra("outputX", 1); // 剪切的宽高比为1：2
intent.putExtra("outputY", 2);
intent.putExtra("aspectX", 20); // 保存图片的宽和高
intent.putExtra("aspectY", 40);
intent.putExtra("scale", true);
intent.putExtra("noFaceDetection", true);
intent.putExtra("output", Uri.parse("file:///mnt/sdcard/temp"));
startActivityForResult(intent, 0);

//=================================================================

//11.打开Google Market
// 打开Google Market直接进入该程序的详细页面
Uri uri = Uri.parse("market://details?id=" + "com.demo.app");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);

//=================================================================

//12.进入手机设置界面:
// 进入无线网络设置界面（其它可以举一反三）
Intent intent = new
Intent(android.provider.Settings.ACTION_WIRELESS_SETTINGS);
startActivityForResult(intent, 0);

//=================================================================

//13.安装apk:
Uri installUri = Uri.fromParts("package", "xxx", null);
returnIt = new Intent(Intent.ACTION_PACKAGE_ADDED, installUri);

//=================================================================

//14.卸载apk:
Uri uri = Uri.fromParts("package", strPackageName, null);
Intent it = new Intent(Intent.ACTION_DELETE, uri);
startActivity(it);

//=================================================================

//15.发送附件:
Intent it = new Intent(Intent.ACTION_SEND);
```

```
it.putExtra(Intent.EXTRA_SUBJECT, "The email subject text");
it.putExtra(Intent.EXTRA_STREAM, "file:///sdcard/eoe.mp3");
sendIntent.setType("audio/mp3");
startActivity(Intent.createChooser(it, "Choose Email Client"));

//===========================================================

//16.进入联系人页面:
Intent intent = new Intent();
intent.setAction(Intent.ACTION_VIEW);
intent.setData(People.CONTENT_URI);
startActivity(intent);

//===========================================================


//17.查看指定联系人:
Uri personUri = ContentUris.withAppendedId(People.CONTENT_URI,
info.id);//info.id联系人ID
Intent intent = new Intent();
intent.setAction(Intent.ACTION_VIEW);
intent.setData(personUri);
startActivity(intent);

//===========================================================

//18.调用系统编辑添加联系人（高版本SDK有效）：
Intent it = newIntent(Intent.ACTION_INSERT_OR_EDIT);
it.setType("vnd.android.cursor.item/contact");
//it.setType(Contacts.CONTENT_ITEM_TYPE);
it.putExtra("name","myName");
it.putExtra(android.provider.Contacts.Intents.Insert.COMPANY,
"organization");
it.putExtra(android.provider.Contacts.Intents.Insert.EMAIL,"email");

it.putExtra(android.provider.Contacts.Intents.Insert.PHONE,"homePhone"
);
it.putExtra(android.provider.Contacts.Intents.Insert.SECONDARY_PHONE,"
mobilePhone");
it.putExtra(
android.provider.Contacts.Intents.Insert.TERTIARY_PHONE,"workPhone");

it.putExtra(android.provider.Contacts.Intents.Insert.JOB_TITLE,"title"
);
startActivity(it);

//===========================================================

//19.调用系统编辑添加联系人（全有效）：
Intent intent = newIntent(Intent.ACTION_INSERT_OR_EDIT);
intent.setType(People.CONTENT_ITEM_TYPE);
intent.putExtra(Contacts.Intents.Insert.NAME, "My Name");
intent.putExtra(Contacts.Intents.Insert.PHONE, "+1234567890");
intent.putExtra(Contacts.Intents.Insert.PHONE_TYPE,Contacts.PhonesColu
mns.TYPE_MOBILE);
intent.putExtra(Contacts.Intents.Insert.EMAIL, "com@com.com");
intent.putExtra(Contacts.Intents.Insert.EMAIL_TYPE,
Contacts.ContactMethodsColumns.TYPE_WORK);
```

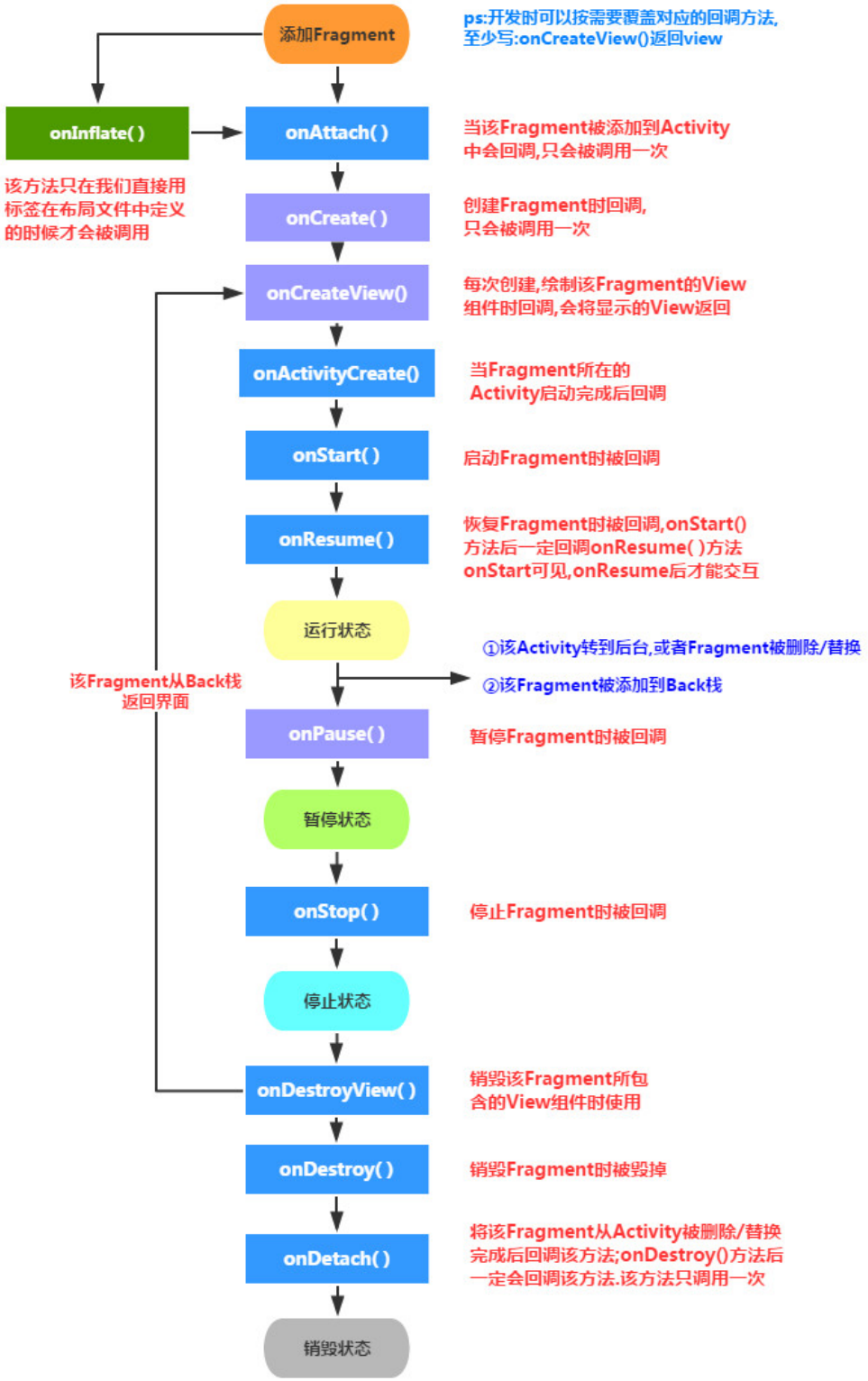
```
startActivity(intent);

//==========================================================

//20.打开另一程序
Intent i = new Intent();
ComponentName cn = new ComponentName("com.example.jay.test",
"com.example.jay.test.MainActivity");
i.setComponent(cn);
i.setAction("android.intent.action.MAIN");
startActivityForResult(i, RESULT_OK);

//==========================================================

//21.打开录音机
Intent mi = new Intent(Media.RECORD_SOUND_ACTION);
startActivity(mi);

//==========================================================

//22.从google搜索内容
Intent intent = new Intent();
intent.setAction(Intent.ACTION_WEB_SEARCH);
intent.putExtra(SearchManager.QUERY,"searchString")
startActivity(intent);

//==========================================================
```

1.6 Fragment（碎片）

> 我们可以把他看成一个小型的Activity，又称Activity片段！想想，如果一个很大的界面，我们 就一个布局，写起界面来会有多麻烦，而且如果组件多的话是管理起来也很麻烦！而使用Fragment我们可以把屏幕划分成几块，然后进行分组，进行一个模块化的管理！从而可以更加方便的在 运行过程中动态地更新Activity的用户界面！另外Fragment并不能单独使用，他需要嵌套在Activity中使用，尽管他拥有自己的生命周期，但是还是会受到宿主Activity的生命周期的影响，比如Activity 被destory销毁了，他也会跟着销毁

- Fragment的生命周期图

Fragment的生命周期图



①Activity加载Fragment的时候,依次调用下面的方法: onAttach -> onCreate -> onCreateView -> onActivityCreated -> onStart ->onResume

②当我们弄出一个悬浮的对话框风格的Activity,或者其他,就是让Fragment所在的Activity可见,但不获得焦点 onPause

③当对话框关闭,Activity又获得了焦点: onResume

④当我们替换Fragment,并调用addToBackStack()将他添加到Back栈中 onPause -> onStop -> onDestoryView ！！注意,此时的Fragment还没有被销毁哦!!!

⑤当我们按下键盘的回退键，Fragment会再次显示出来: onCreateView -> onActivityCreated -> onStart -> onResume

⑥如果我们替换后,在事务commit之前没有调用addToBackStack()方法将 Fragment添加到back栈中的话;又或者退出了Activity的话,那么Fragment将会被完全结束， Fragment会进入销毁状态 onPause -> onStop -> onDestoryView -> onDestory -> onDetach

addToBackStack()方法的作用：当移除或替换一个Fragment并向返回栈添加事务时，系统会停止（而非销毁）移除的Fragment。如果用户执行回退操作进行Fragment的恢复，该Fragment将重新启动。如果不向返回栈添加事务，则系统会在移除或替换Fragment时将其销毁。

- 使用V4包还是app包下面的Fragment

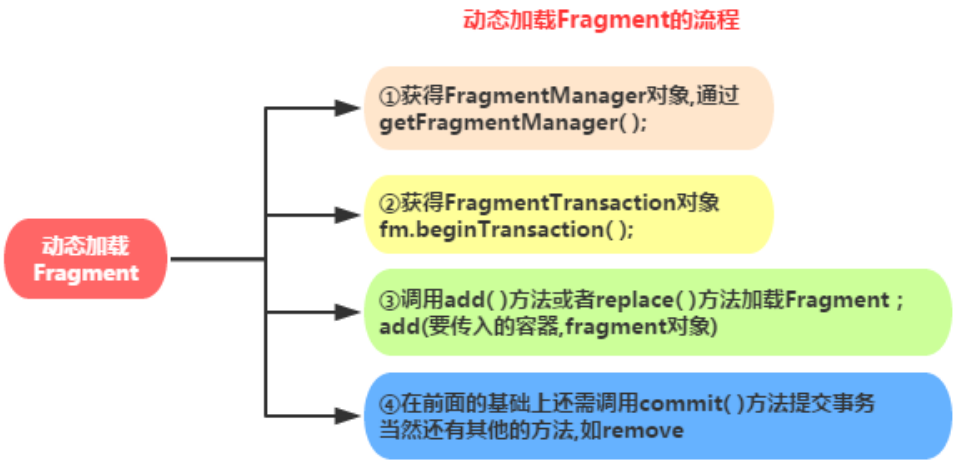
其实都可以，前面说过Fragment是Android 3.0(API 11)后引入的，那么如果开发的app需要 在3.0以下的版本运行呢?比如还有一点点市场份额的2.3!于是乎,v4包就这样应运而生了， 而最低可以兼容到1.6版本！至于使用哪个包看你的需求了,现在3.0下手机市场份额其实已经不多了,随街都是4.0以上的，6.0十月份都出了，你说呢...所以这个时候,你可以直接使用app包下的Fragment 然后调用相关的方法，通常都是不会有什么问题的;如果你Fragment用了app包的， FragmentManager和FragmentTransaction都需要是app包的！要么用全部用app,要么全部用v4， 不然可是会报错的哦！当然如果你要自己的app对于低版本的手机也兼容的话,那么就可以选择用v4包！

- 创建一个Fragment

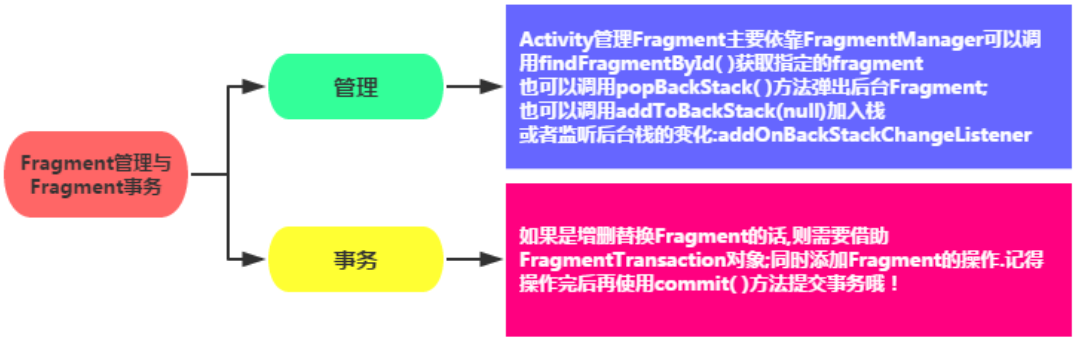
1. 静态加载Fragment

```
<fragment
        android:id="@+id/myfragment"
        android:name="com.ttit.core.Fragment.MyFragment"
        android:layout_width="match_parent"
        android:layout_height="300dp" />

ps: 必须添加id属性
```

2. 动态加载Fragment



- Fragment管理与Fragment事务



- Fragment与Activity的交互