[第一行代码（第一版） 1](#_Toc476515871)

[四大组件 1](#_Toc476515872)

[LogCat 1](#_Toc476515873)

[隐藏标题栏 2](#_Toc476515874)

[Toast 2](#_Toc476515875)

[Intent 2](#_Toc476515876)

[使用隐式 Intent 3](#_Toc476515877)

[向下一个活动传递数据 4](#_Toc476515878)

[返回数据给上一个活动 4](#_Toc476515879)

[活动状态 5](#_Toc476515880)

[活动的生存期 5](#_Toc476515881)

[活动被回收了怎么办 7](#_Toc476515882)

[活动的启动模式 8](#_Toc476515883)

[FIREBASE 9](#_Toc476515884)

[Firebas简介 9](#_Toc476515885)

[Firebase Analytics 9](#_Toc476515886)

[开发 10](#_Toc476515887)

[增长 10](#_Toc476515888)

[获利 10](#_Toc476515889)

# 第一行代码（第一版）

## 四大组件

Android系统四大组件分别是活动（Activity）、服务（Service）、广播接收器（Broadcast

Receiver）和内容提供器（ContentProvider）。其中活动是所有 Android 应用程序的门面，

凡是在应用中你看得到的东西，都是放在活动中的。而服务就比较低调了，你无法看到

它，但它会一直在后台默默地运行，即使用户退出了应用，服务仍然是可以继续运行的。

广播接收器可以允许你的应用接收来自各处的广播消息，比如电话、短信等，当然你的

应用同样也可以向外发出广播消息。内容提供器则为应用程序之间共享数据提供了可

能，比如你想要读取系统电话簿中的联系人，就需要通过内容提供器来实现。

## LogCat

1. Log.v()

这个方法用于打印那些最为琐碎的，意义最小的日志信息。对应级别 verbose，是

Android 日志里面级别最低的一种。

2. Log.d()

这个方法用于打印一些调试信息，这些信息对你调试程序和分析问题应该是有帮助

的。对应级别 debug，比 verbose 高一级。

3. Log.i()

这个方法用于打印一些比较重要的数据，这些数据应该是你非常想看到的，可以帮

你分析用户行为的那种。对应级别 info，比 debug高一级。

4. Log.w()

这个方法用于打印一些警告信息，提示程序在这个地方可能会有潜在的风险，最好

去修复一下这些出现警告的地方。对应级别 warn，比 info 高一级。

5. Log.e()

这个方法用于打印程序中的错误信息，比如程序进入到了 catch 语句当中。当有错

误信息打印出来的时候，一般都代表你的程序出现严重问题了，必须尽快修复。对应级

别 error，比 warn 高一级。

## 隐藏标题栏

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

其中 requestWindowFeature(Window.FEATURE\_NO\_TITLE)的意思就是不在活动中显示

标题栏，注意这句代码一定要在 setContentView()之前执行，不然会报错。

## Toast

Toast的用法非常简单，通过静态方法 makeText()创建出一个 Toast对象，然后调用 show()将 Toast 显示出来就可以了。这里需要注意的是，makeText()方法需要传入三个参数。第一个参数是 Context，也就是 Toast 要求的上下文，由于活动本身就是一个 Context 对象，因此这里直接传入FirstActivity.this即可。第二个参数是Toast显示的文本内容，第三个参数是Toast显示的时长，有两个内置常量可以选择 Toast.LENGTH\_SHORT和 Toast.LENGTH\_LONG。

## Intent

Intent 是 Android 程序中各组件之间进行交互的一种重要方式，它不仅可以指明当前组件想要执行的动作，还可以在不同组件之间传递数据。Intent 一般可被用于启动活动、启动服务、以及发送广播等场景，由于服务、广播等概念你暂时还未涉及，那么本章我们的目光无疑就锁定在了启动活动上面。

Intent 的用法大致可以分为两种，显式 Intent 和隐式 Intent，Intent有多个构造函数的重载，其中一个是 Intent(Context packageContext, Class<?> cls)。这个构造函数接收两个参数，第一个参数 Context 要求提供一个启动活动的上下文，第二个参数 Class 则是指定想要启动的目标活动，通过这个构造函数就可以构建出 Intent 的“意图”。

Activity 类中提供了一个 startActivity()方法，这个方法是专门用于启动活动的，它接收一个Intent参数，这里我们将构建好的Intent传入startActivity()方法就可以启动目标活动了。

## 使用隐式 Intent

相比于显式 Intent，隐式 Intent 则含蓄了许多，它并不明确指出我们想要启动哪一个活动，而是指定了一系列更为抽象的 action和 category等信息，然后交由系统去分析这个 Intent，并帮我们找出合适的活动去启动。

<activity android:name=".SecondActivity" >

<intent-filter>

<action android:name="com.example.activitytest.ACTION\_START" />

<category android:name="android.intent.category.DEFAULT" />

</intent-filter>

</activity>

在<action>标签中我们指明了当前活动可以响应 com.example.activitytest.ACTION\_START这个 action，而<category>标签则包含了一些附加信息，更精确地指明了当前的活动能够响应的 Intent 中还可能带有的 category。只有<action>和<category>中的内容同时能够匹配上 Intent 中指定的 action 和 category 时，这个活动才能响应该 Intent。

使用隐式 Intent，我们不仅可以启动自己程序内的活动，还可以启动其他程序的活动，这使得 Android 多个应用程序之间的功能共享成为了可能。比如说你的应用程序中需要展示一个网页，这时你没有必要自己去实现一个浏览器（事实上也不太可能），而是只需要调用系统的浏览器来打开这个网页就行了。

Intent intent = new Intent(Intent.ACTION\_VIEW);

intent.setData(Uri.parse("http://www.baidu.com"));

这里我们首先指定了 Intent 的 action 是 Intent.ACTION\_VIEW，这是一个 Android 系统内置的动作，其常量值为 android.intent.action.VIEW。然后通过 Uri.parse()方法，将一个网址字符串解析成一个 Uri 对象，再调用 Intent的 setData()方法将这个 Uri 对象传递进去。

我们还可以在<intent-filter>标签中再配置一个<data>标签，用于更精确地指定当前活动能够响应什么类型的数据。<data>标签中主要可以配置以下内容。

1. android:scheme

用于指定数据的协议部分，如上例中的 http 部分。

2. android:host

用于指定数据的主机名部分，如上例中的 www.baidu.com部分。

3. android:port

用于指定数据的端口部分，一般紧随在主机名之后。

4. android:path

用于指定主机名和端口之后的部分，如一段网址中跟在域名之后的内容。

5. android:mimeType

用于指定可以处理的数据类型，允许使用通配符的方式进行指定。

只有<data>标签中指定的内容和 Intent 中携带的 Data完全一致时，当前活动才能够响应该 Intent。不过一般在<data>标签中都不会指定过多的内容，如上面浏览器示例中，其实只需要指定 android:scheme 为 http，就可以响应所有的 http 协议的 Intent 了。

除了 http 协议外，我们还可以指定很多其他协议，比如 geo 表示显示地理位置、tel 表示拨打电话。

Intent intent = new Intent(Intent.ACTION\_DIAL);

intent.setData(Uri.parse("tel:10086"));

首先指定了 Intent 的 action 是 Intent.ACTION\_DIAL，这又是一个 Android 系统的内置动作。然后在 data部分指定了协议是 tel，号码是 10086。

## 向下一个活动传递数据

在启动活动时传递数据的思路很简单，Intent中提供了一系列 putExtra()方法的重载，可以把我们想要传递的数据暂存在 Intent 中，启动了另一个活动后，只需要把这些数据再从Intent 中取出就可以了。

String data = "Hello SecondActivity";

Intent intent = new Intent(FirstActivity.this, SecondActivity.class);

intent.putExtra("extra\_data", data);

startActivity(intent);

这里我们还是使用显式 Intent 的方式来启动 SecondActivity，并通过 putExtra()方法传递了一个字符串。注意这里 putExtra()方法接收两个参数，第一个参数是键，用于后面从 Intent中取值，第二个参数才是真正要传递的数据。

在 SecondActivity 中将传递的数据取出:

Intent intent = getIntent();

String data = intent.getStringExtra("extra\_data");

首先可以通过 getIntent()方法获取到用于启动 SecondActivity 的 Intent，然后调用getStringExtra()方法，传入相应的键值，就可以得到传递的数据了。这里由于我们传递的是字符串，所以使用 getStringExtra()方法来获取传递的数据，如果传递的是整型数据，则使用getIntExtra()方法，传递的是布尔型数据，则使用 getBooleanExtra()方法，以此类推。

## 返回数据给上一个活动

既然可以传递数据给下一个活动，那么能不能够返回数据给上一个活动呢？答案是肯定的。不过不同的是，返回上一个活动只需要按一下 Back 键就可以了，并没有一个用于启动活动 Intent 来传递数据。通过查阅文档你会发现，Activity 中还有一个 startActivityForResult()方法也是用于启动活动的，但这个方法期望在活动销毁的时候能够返回一个结果给上一个活

动。

startActivityForResult()方法接收两个参数，第一个参数还是 Intent，第二个参数是请求

码，用于在之后的回调中判断数据的来源。

Intent intent = new Intent();

intent.putExtra("data\_return", "Hello FirstActivity");

setResult(RESULT\_OK, intent);

finish();

可以看到，我们还是构建了一个 Intent，只不过这个 Intent 仅仅是用于传递数据而已，它没有指定任何的“意图”。紧接着把要传递的数据存放在 Intent 中，然后调用了 setResult()方法。这个方法非常重要，是专门用于向上一个活动返回数据的。setResult()方法接收两个参数，第一个参数用于向上一个活动返回处理结果，一般只使用 RESULT\_OK 或RESULT\_CANCELED 这两个值，第二个参数则是把带有数据的 Intent 传递回去，然后调用了 finish()方法来销毁当前活动。由于我们是使用 startActivityForResult()方法来启动 SecondActivity 的，在 SecondActivity被销毁之后会回调上一个活动的 onActivityResult()方法，因此我们需要在 FirstActivity 中重写这个方法来得到返回的数据。

onActivityResult()方法带有三个参数，第一个参数 requestCode，即我们在启动活动时传入的请求码。第二个参数 resultCode，即我们在返回数据时传入的处理结果。第三个参数 data，即携带着返回数据的 Intent。由于在一个活动中有可能调用 startActivityForResult()方法去启动很多不同的活动，每一个活动返回的数据都会回调到 onActivityResult()这个方法中，因此我们首先要做的就是通过检查 requestCode 的值来判断数据来源。

## 活动状态

每个活动在其生命周期中最多可能会有四种状态。

1. 运行状态

当一个活动位于返回栈的栈顶时，这时活动就处于运行状态。系统最不愿意回收的就是处于运行状态的活动，因为这会带来非常差的用户体验。

2. 暂停状态

当一个活动不再处于栈顶位置，但仍然可见时，这时活动就进入了暂停状态。你可能会觉得既然活动已经不在栈顶了，还怎么会可见呢？这是因为并不是每一个活动都会占满整个屏幕的，比如对话框形式的活动只会占用屏幕中间的部分区域，你很快就会在后面看到这种活动。处于暂停状态的活动仍然是完全存活着的，系统也不愿意去回收这种活动（因为它还是可见的，回收可见的东西都会在用户体验方面有不好的影响），只有在内存极低的情况下，系统才会去考虑回收这种活动。

3. 停止状态

当一个活动不再处于栈顶位置，并且完全不可见的时候，就进入了停止状态。系统仍然会为这种活动保存相应的状态和成员变量，但是这并不是完全可靠的，当其他地方需要内存时，处于停止状态的活动有可能会被系统回收。

4. 销毁状态

当一个活动从返回栈中移除后就变成了销毁状态。系统会最倾向于回收处于这种状态的活动，从而保证手机的内存充足。

## 活动的生存期

Activity 类中定义了七个回调方法，覆盖了活动生命周期的每一个环节。

1. onCreate()

这个方法你已经看到过很多次了，每个活动中我们都重写了这个方法，它会在活动第一次被创建的时候调用。你应该在这个方法中完成活动的初始化操作，比如说加载布局、绑定事件等。

2. onStart()

这个方法在活动由不可见变为可见的时候调用。

3. onResume()

这个方法在活动准备好和用户进行交互的时候调用。此时的活动一定位于返回栈的栈顶，并且处于运行状态。

4. onPause()

这个方法在系统准备去启动或者恢复另一个活动的时候调用。我们通常会在这个方法中将一些消耗 CPU 的资源释放掉，以及保存一些关键数据，但这个方法的执行速度一定要快，不然会影响到新的栈顶活动的使用。

5. onStop()

这个方法在活动完全不可见的时候调用。它和 onPause()方法的主要区别在于，如果启动的新活动是一个对话框式的活动，那么 onPause()方法会得到执行，而 onStop()方法并不会执行。

6. onDestroy()

这个方法在活动被销毁之前调用，之后活动的状态将变为销毁状态。

7. onRestart()

这个方法在活动由停止状态变为运行状态之前调用，也就是活动被重新启动了。以上七个方法中除了 onRestart()方法，其他都是两两相对的，从而又可以将活动分为三种生存期。

1. 完整生存期

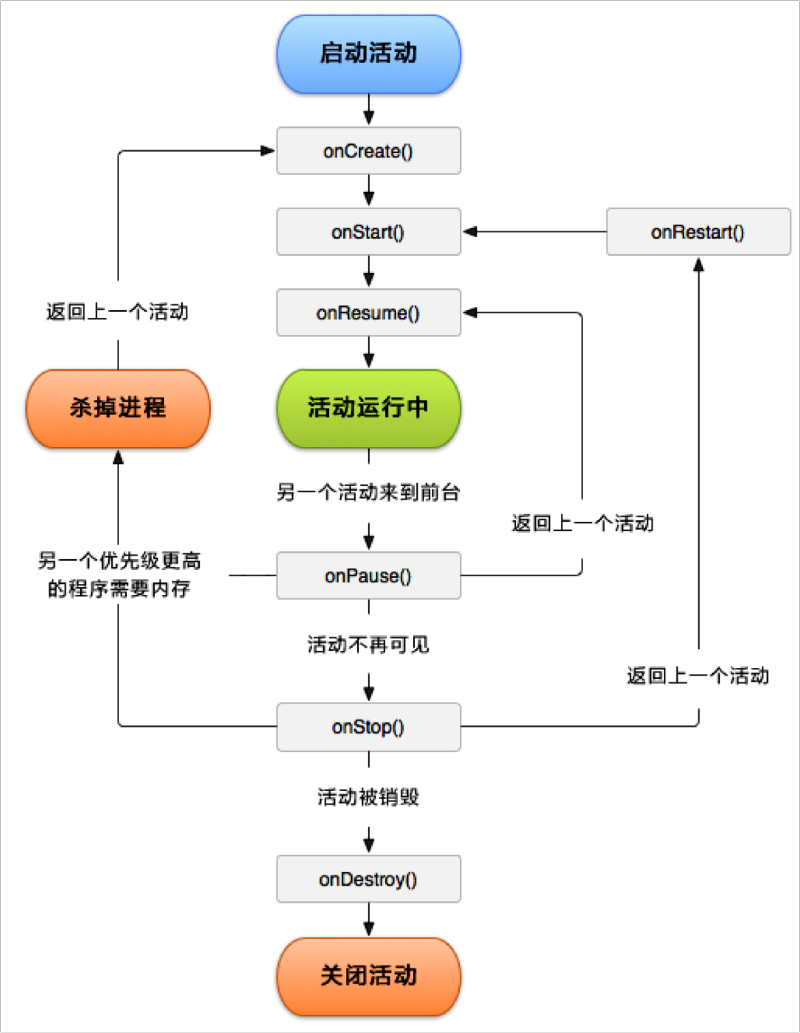
活动在 onCreate()方法和 onDestroy()方法之间所经历的，就是完整生存期。一般情况下，一个活动会在 onCreate()方法中完成各种初始化操作，而在 onDestroy()方法中完成释放内存的操作。

2. 可见生存期

活动在 onStart()方法和 onStop()方法之间所经历的，就是可见生存期。在可见生存期内，活动对于用户总是可见的，即便有可能无法和用户进行交互。我们可以通过这两个方法，合理地管理那些对用户可见的资源。比如在 onStart()方法中对资源进行加载，而在onStop()方法中对资源进行释放，从而保证处于停止状态的活动不会占用过多内存。

3. 前台生存期

活动在 onResume()方法和 onPause()方法之间所经历的，就是前台生存期。在前台生存期内，活动总是处于运行状态的，此时的活动是可以和用户进行相互的，我们平时看到和接触最多的也这个状态下的活动。



## 活动被回收了怎么办

Activity 中还提供了一个 onSaveInstanceState()回调方法，这个方法会保证一定在活动被回收之前调用，因此我们可以通过这个方法来解决活动被回收时临时数据得不到保存的问题。onSaveInstanceState()方法会携带一个 Bundle 类型的参数，Bundle 提供了一系列的方法用于保存数据，比如可以使用 putString()方法保存字符串，使用 putInt()方法保存整型数据，以此类推。每个保存方法需要传入两个参数，第一个参数是键，用于后面从 Bundle中取值，第二个参数是真正要保存的内容。

一直使用的 onCreate()方法其实也有一个 Bundle类型的参数。这个参数在一般情况下都是null，但是当活动被系统回收之前有通过 onSaveInstanceState()方法来保存数据的话，这个参数就会带有之前所保存的全部数据，我们只需要再通过相应的取值方法将数据取出即可。

if (savedInstanceState != null) {

String tempData = savedInstanceState.getString("data\_key");

Log.d(TAG, tempData);

}

取出值之后再做相应的恢复操作就可以了，使用 Bundle 来保存和取出数据是不是有些似曾相识呢？没错！我们在使用 Intent 传递数据时也是用的类似的方法。这里跟你提醒一点，Intent 还可以结合Bundle 一起用于传递数据的，首先可以把需要传递的数据都保存在 Bundle 对象中，然后再将 Bundle 对象存放在 Intent 里。到了目标活动之后先从 Intent 中取出 Bundle，再从 Bundle中一一取出数据。

## 活动的启动模式

在实际项目中我们应该根据特定的需求为每个活动指定恰当的启动模式。启动模式一共有四种，分别是 standard、singleTop、singleTask 和 singleInstance，可以在 AndroidManifest.xml 中通 过给<activity> 标签 指定android:launchMode属性来选择启动模式。

tandard 是活动默认的启动模式，在不进行显式指定的情况下，所有活动都会自动使用这种启动模式。在 standard 模式（即默认情况）下，每当启动一个新的活动，它就会在返回栈中入栈，并处于栈顶的位置。对于使用standard 模式的活动，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。

singleTop可能在有些情况下，你会觉得 standard 模式不太合理。活动明明已经在栈顶了，为什么再次启动的时候还要创建一个新的活动实例呢？别着急，这只是系统默认的一种启动模式而已，你完全可以根据自己的需要进行修改，比如说使用 singleTop 模式。当活动的启动模式指定为 singleTop，在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。

singleTask使用 singleTop 模式可以很好地解决重复创建栈顶活动的问题，如果该活动并没有处于栈顶的位置，还是可能会创建多个活动实例的。那么有没有什么办法可以让某个活动在整个应用程序的上下文中只存在一个实例呢？这就要借助singleTask模式来实现了。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。

singleInstance模式应该算是四种启动模式中最特殊也最复杂的一个了，不同于以上三种启动模式，指定为 singleInstance模式的活动会启用一个新的返回栈来管理这个活动（其实如果 singleTask模式指定了不同的 taskAffinity，也会启动一个新的返回栈）。那么这样做有什么意义呢？想象以下场景，假设我们的程序中有一个活动是允许其他程序调用的，如果我们想实现其他程序和我们的程序可以共享这个活动的实例，应该如何实现呢？使用前面三种启动模式肯定是做不到的，因为每个应用程序都会有自己的返回栈，同一个活动在不同的返回栈中入栈时必然是创建了新的实例。而使用singleInstance模式就可以解决这个问题，在这种模式下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，都共用的同一个返回栈，也就解决了共享活动实例的问题。

# FIREBASE

Firebas简介：

提供了各种工具，帮助您开发高品质应用，扩大用户群，并赚取更多收益。我们包揽了基础性工作，这样您就可以通过业务获得收入，并专注于您的用户。

Firebase Analytics ：

Firebase 的核心是 Firebase Analytics，一款免费并且没有任何限制的解决方案。 Analytics 跨越 Firebase 功能进行集成，并且可以无限制地向您提供多达 500 个不同的事件的报告，您可以利用 Firebase SDK 定义这些事件。从单一信息中心查看用户行为和衡量行为特性。 Firebase Analytics 报告可帮助您清晰地了解您的用户的行为方式，让您可以针对应用营销和性能优化制定明智的决策。

主要功能：

（1）无限制报告：Firebase Analytics 可提供多达 500 种不同事件的无限制报告。

（2）目标设备细分：可以基于设备数据、自定义事件或用户属性在 Firebase console 中定义自定义目标设备。 在锁定新功能或通知时，这些目标设备可以与其他 Firebase 功能结合使用。

如何工作：

Firebase Analytics 可帮助您了解人们使用您的 iOS 或 Android 应用的方式。 SDK 自动捕获大量的事件和用户属性，同时也允许您定义您自己的自定义事件以衡量对您的业务具有独特意义的内容。

捕获数据后，可在 Firebase 控制台的信息中心中找到它们。 通过此信息中心，您可以详细了解您的数据 — 从摘要数据（如活跃用户数和人口统计）到更详细的数据（如确定最畅销的商品）。

Firebase Analytics 还集成了许多其他 Firebase 功能。 例如，它自动记录与您的 Firebase Notifications 对应的事件，并提供有关每个广告系列的影响的报告。

Firebase Analytics 可帮助您了解您的用户的行为方式，以便您可以就如何推广您的应用制定明智的决策。 查看您的广告系列在自然渠道和付费渠道的绩效，以了解哪个方法对于吸引高价值用户最有效。

如果您需要执行自定义分析或将您的数据与其他源进行结合，您可以将您的 Analytics 数据链接至 BigQuery，其允许您进行更复杂的分析，如查询大型数据集和加入多个数据源。

与其他服务集成：

1. Big Query：将您的 Firebase Analytics 应用链接至 Big Query，在该服务中您可以对您的整个 Analytics 数据集进行自定义分析，并导入其他数据源。
2. Firebase Crash Reporting ：Firebase Analytics 记录每个崩溃的事件，以便您可以了解不同版本或地区的崩溃率，从而洞悉哪些用户受到影响。 您还可以为经历过多次崩溃的用户创建目标设备，并通过针对该设备的 Firebase Notifications 进行响应。
3. Firebase Notifications：Firebase Analytics 自动记录与您的 Firebase Notifications 对应的事件，并提供有关每个广告系列的影响的报告。
4. Firebase Remote Config：使用 Firebase Analytics 目标设备定义针对不同的目标设备更改应用的行为和外观，无需分发应用的多个版本。
5. Google 跟踪代码管理器：通过将 Google 跟踪代码管理器 与 Firebase Analytics 集成，在分发您的应用后，您可以在远程通过网络界面管理您的 Firebase Analytics 实现。

实现路径：

1. 将您的应用连接到 Firebase。

Analytics 很容易入门。只需将 Firebase SDK 添加到新的或现有应用，即会自动开始数据收集。 数小时内，您就可以在 Firebase console 中查看分析数据。

1. 记录自定义数据

您可以使用 Analytics 记录对您的应用有意义的自定义事件，如电子商务采购或成就。

1. 创建目标设备

您可以在 Firebase console 中定义对您很重要的目标设备。

1. 目标设备

使用您的自定义目标设备锁定消息、促销活动或使用其他 Firebase 功能的新应用功能，如 Notifications以及 Remote Config。

开发：专心构建更好的应用，将具体操作留给我们来完成。节省宝贵的开发时间，交付高品质、无缺陷的应用。

增长：在合适的时间赢得和吸引合适的用户。不靠碰运气，实现精准稳健增长。

获利：借助丰富且引人入胜的广告通过您的应用获利。