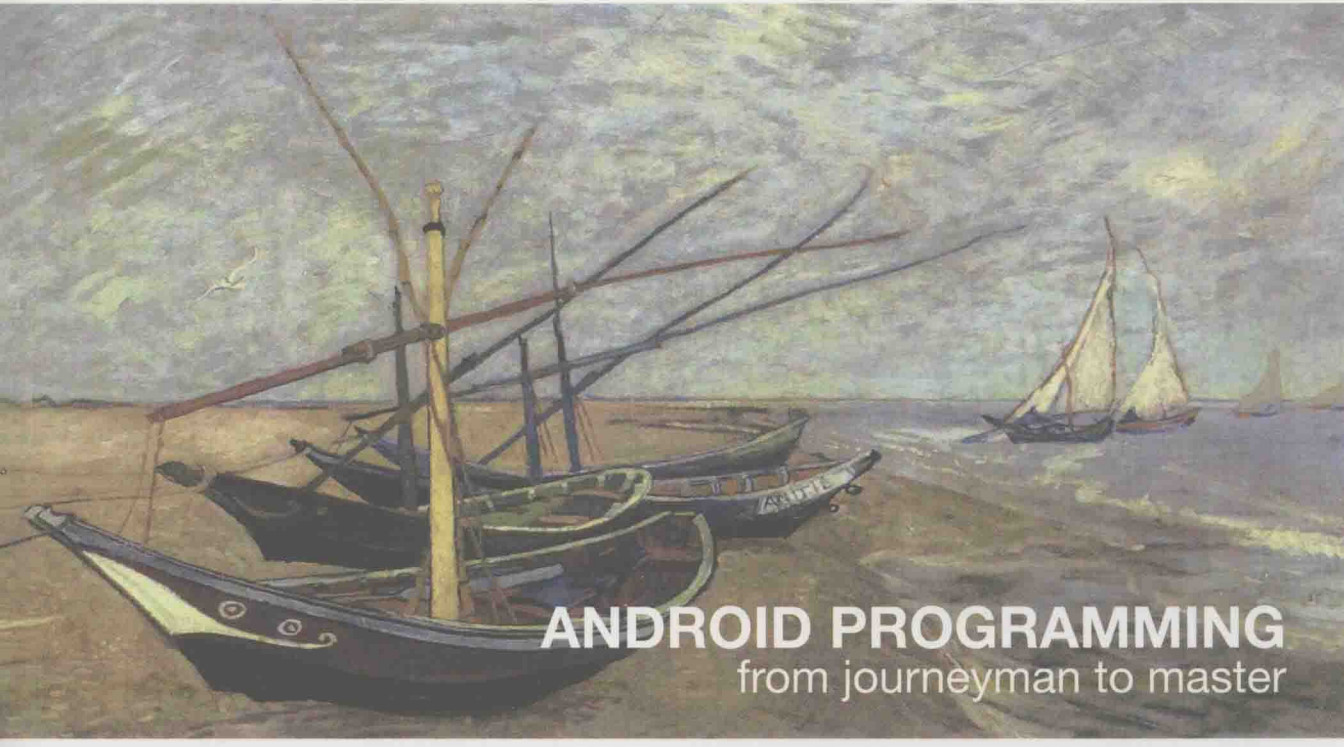
ANDROID
BASE ANDROID 5.0 AND OVER

Android开发进阶
从小工到专家

◆ 何红辉 著



中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

ANDROID
BASE ANDROID 5.0 AND OVER

Android开发进阶
从小工到专家

阿红辉 著

ANDROID PROGRAMMING
from journeyman to master

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Android开发进阶 : 从小工到专家 / 何红辉著. --
北京 : 人民邮电出版社, 2016.2
ISBN 978-7-115-41591-2

Ⅰ. ①A… Ⅱ. ①何… Ⅲ. ①移动终端－应用程序－
程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2016)第016330号

内 容 提 要

本书是一本专门介绍 Android 开发的图书。书中首先对 Android 开发的核心知识点进行深入讲解，然后介绍单元测试、代码规范、版本控制、重构、架构等重要的知识，使得读者在深入掌握技术的同时也帮助他们开阔眼界，且能够以更专业的方式设计应用软件，完成从只会实现功能的初级程序员到软件工程师、设计师的转变。

本书的主要内容为：构成 Android 系统基石的四大组件、创造出丰富多彩的 UI 设计的控件、保证 App 流畅的多线程开发、必知必会的 HTTP 网络请求应用、数据存储的核心 SQLite 数据库、让程序更优秀的性能优化、让程序更整洁的代码规范、管理程序的好帮手 Git 版本控制，以及需要掌握的高级技术，如单元测试、六大原则与设计模式、重构和综合实战等。

本书适合 Android 开发初学者、程序员学习，也适合作为大中专院校相关专业的师生用书和培训学校的教材。

◆ 著　　何红辉
责任编辑　张　涛
责任印制　张佳莹　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：24.5
字数：660 千字　　2016 年 2 月第 1 版
印数：1－3 000 册　　2016 年 2 月北京第 1 次印刷

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前　言

为什么写这本书

写这本书的念头由来已久了。也许是从我打算写《Android 源码设计模式解析与实战》那时起就萌生了这个念头，因为设计模式属于仅次于架构之下的局部战术，阅读这类书籍能够让具备一定工作经验的开发人员提升自己的设计能力，构建更灵活的软件。但是，对于初、中级工程师而言，最重要的还是在于基础知识以及知识广度的掌握上。因此，在《Android 源码设计模式解析与实战》交稿之后，我就立即开始了本书的写作之旅。

从单位的面试经历和与开发群中网友的交流中，我发现很多有一定工作经验的开发人员对于 Android 的基础知识都还只停留在“会用”的阶段，而对于其基本原理一概不知，以至于工作多年之后依旧停留在很表面的层次。这样的知识结构的程序员往往是一旦开发的系统出现问题或者需要优化时就不能应对了。因此，仔细阅读一本深入讲述 Android 核心开发知识点的书是很有必要的。

目前，图书市场上关于 Android 的入门书籍大多是覆盖整个 Android 开发知识体系，这类书籍的特点是讲解的知识面多，也正是这个原因使得这类书籍缺乏深度，往往只是点到即止。例如，关于网络请求的技术，通常只讲解如何发送一个 GET 请求，但是，对于 HTTP 原理不会涉及，这使得很多读者在定制一些请求时根本无从下手，如上传图片、参数格式为 Json 等。

另一个问题就是，很多开发人员即使从业多年，可能都不知道什么是单元测试，不知道重构、面向对象基本原则，这使得他们的代码耦合度可能很高，难以测试和维护，这样带来的后果就是质量没法保证，随着时间的推移系统逐渐“腐化”。因此，读一本讲述设计软件的书也是必要的。

本书的目的就是解决上述两个问题，首先对 Android 开发的核心知识点进行深入讲解，然后介绍单元测试、代码规范、版本控制、重构、架构等重要知识点，使得读者在深入技术的同时开阔眼界，能够以更专业的方式设计应用软件，帮助读者完成从只会实现功能的“码农”到软件工程师、设计师的过渡。

本书的特色

本书主要分为 3 部分，第一部分是前 6 章，在第一部分中深入讲解了 Android 开发过程中的核心知识点，包括 View 与动画、多线程、网络、数据库、性能优化，使得读者深入了解开发中最为重要的知识；第二部分是第 7~11 章，涵盖的内容包括代码规范、单元测试、版本控制、OOP 与模式、重构等内容，从代码规范化、专业化的角度着手，开阔读者的眼界，使读者具备构建低耦合、

灵活性强的应用软件的基本能力；最后一部分是第 12 章，在第 12 章中通过一个完整的示例，演示了如何把一个充满问题的应用软件逐步演化为低耦合、清晰、可测试的实现过程，在其中展示了常见的重构手法、测试手段，使读者从真实的示例中汲取知识与经验，提升技术与设计能力，绕过编程中的诸多陷阱。

当然，书中的知识点很多都只是做了部分讲解，起到一个抛砖引玉的作用，因此，如果需要更深入地了解各领域的知识，希望读者阅读其他专业书籍。

面向的读者

本书面向的读者为初、中、高级 Android 工程师。本书的定位是学习 Android 开发的第二本书，因此，阅读的前提是读者需要有一定的 Android 开发知识。在阅读完本书之后，读者还可以选择《Android 群英传》《Android 开发艺术探索》《Android 源码设计模式解析与实战》等书进行更深入地学习，从更深、更高的层次提升自己，完成从“码农”到专家的蜕变。

如何阅读本书

本书从整体结构上分为 3 部分，分别为 Android 核心开发知识、规范化与专业化开发基本知识、实战示例。初、中级工程师建议阅读全书，高级工程师可以选择自己感兴趣的部分进行阅读。实战示例部分需要第二部分的知识，因此，在阅读最后一章时，如果你学习了第二部分的知识，那么理解效果会更好。判定你是否需要阅读某个章节的标准是，当你看到标题时是否对这个知识点了然于心，如果答案是否定的，那么阅读该章节还是很有必要的。当然，通读全书自然是最好的选择。

“纸上得来终觉浅，绝知此事要躬行”，这放到任何一本书中都适用。因此，阅读本书时建议重新完成书中的示例，然后进行思考，从中体会为什么要这样做，这样做得到的好处是什么。读书、实践、思考结合起来，才会让你在技术道路上跑得更快、更远！

读者反馈

最后需要说明的是，任何一本书籍都难免会有一些错误的地方，因此，我很乐意听到读者关于本书的意见或建议，希望与大家共同进步。读者可以通过发邮件（邮箱地址：simplecoder.h@gmail.com）的方式进行反馈，在这里致以诚挚的谢意。编辑联系邮箱：zhangtao@ptpress.com.cn。

代码下载

本书中的示例代码都托管在 Github，地址为 https://github.com/bboyfeiyu/android_jtm_sourcecode，读者可以通过 Git 进行下载。另外，本书的勘误地址为：https://github.com/bboyfeiyu/android-jtm-issues。如果读者发现了书中的错误，也可以提交到该项目中。

何红辉

于北京

致　谢

在本书的出版过程中得到了很多朋友的帮助，首先要感谢佳星、毕老师、凯子、小雨、文辉等好友的审稿，他们的付出使书中的大部分文字错误都在早期被修正。最重要的是要感谢张涛编辑的信任，在上一本书《Android 源码设计模式解析与实战》的写作过程中，张涛编辑给了我很大的自由空间，对于我的各种问题也是耐心解答，也正是这些原因使我毫不犹豫地再度与张涛编辑合作。最后要感谢我的家人，在我写作的时候给我建议、校稿，在整个过程中给予我很大的支持。

何红辉

于北京

目　录

第 1 章　Android 的构成基石——四大组件

由于本书的目标读者是有一定 Android 基础的开发人员，因此，本章不再介绍 Android 系统的架构、历史等知识，而是直接切入主题，从讲解 Android 的四大组件开始，然后一步一步深入学习开发中的重要知识点，使得我们能够从基本原理层面掌握 Android 开发基础知识。

Android 中最重要的是四大组件，即 Activity、Service、ContentProvider 和 Broadcast。这 4 个组件分工明确，共同构成了可重用、灵活、低耦合的 Android 系统。Activity 负责 UI 元素的加载与页面之间的跳转，代表了一个页面单元；Service 负责与 UI 无关的工作，如在后台执行耗时操作等；ContentProvider 负责存储、共享数据，使得数据可以在多个应用之间共享；Broadcast 则是在各个组件、应用之间进行通信，简化了 Android 开发中的通信问题。

下面就来简单学习一下这四大开发组件。

1.1 Activity

Activity 在应用中的表现就是一个用户界面，它会加载指定的布局文件来显示各种 UI 元素，例如 TextView、Button、ImageView、ListView 等，并且为这些 UI 元素设置事件处理函数，使得用户可以与这些 UI 进行交互。同时，Activity 还可以在不同的 Activity 之间跳转，将不同的页面串连在一起，共同完成特定的操作流程。每个应用都是由一个或者多个 Activity 组成，它是 Android 应用程序中不可缺少的部分。

应用启动时会加载一个默认的 Activity，这个 Activity 在 AndroidManifest.xml 中会被设置为如下 intent-filter：

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

每个 Activity 都有生命周期，在不同的阶段会回调不同的生命周期函数，Activity 的生命周期函数有如下几个。

1. onCreate()

相信这是开发者见过次数最多的函数，我们在创建继承自 Activity 的类时都会默认生成这个函

数。它会在 Activity 第一次被创建时调用，通常会在这个函数中完成 Activity 的初始化操作，如设置布局、初始化视图、绑定事件等。

2. onStart()

这个函数在 Activity 的 onCreate 函数调用之后被调用，此时的 Activity 还处在不可见状态，它的下一个状态就是 Activity 变得可见的时候，也就是这个函数在 Activity 可见之前被调用。

3. onResume()

这个函数在 Activity 变为可见时被调用，执行完 onResume 之后，Activity 就会请求 AMS 渲染它所管理的视图。此时的 Activity 一定位于返回栈的栈顶，并且处于运行状态。

4. onPause()

这个函数在系统准备去启动或者恢复另一个 Activity 时调用，也就是在 Activity 即将从可见状态变为不可见时。我们通常会在这个函数中将一些消耗 CPU 的资源释放掉，以及保存一些关键数据。

5. onStop()

这个函数在 Activity 完全不可见时调用。它和 onPause()函数的主要区别在于，如果新启动的 Activity 是一个对话框式的 Activity，那么 onPause()函数会得到执行，而 onStop() 函数并不会执行。

6. onDestroy()

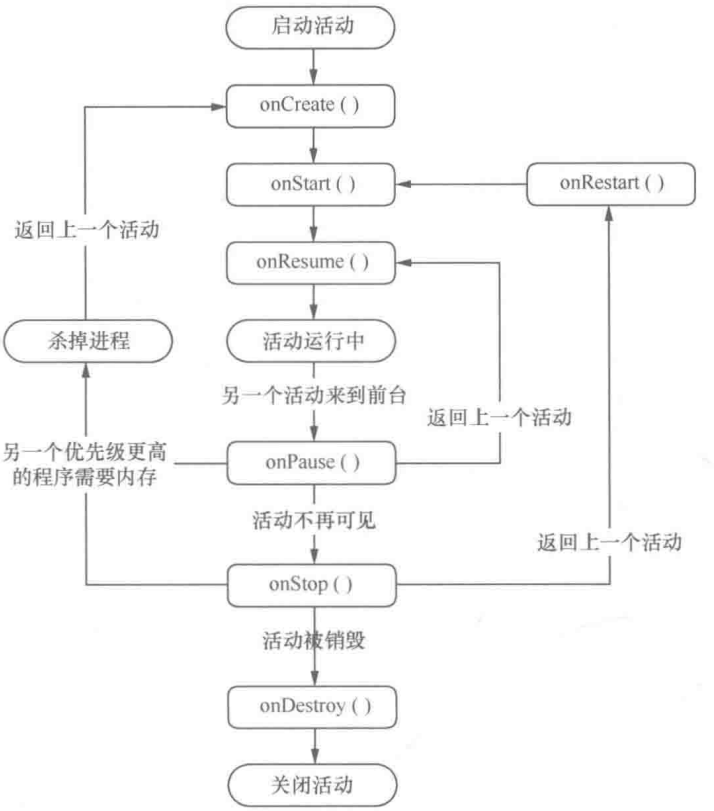
这个函数在 Activity 被销毁之前调用，之后 Activity 的状态将变为销毁状态。

7. onRestart()

这个函数在 Activity 由停止状态重新变为运行状态之前调用，也就是 Activity 被重新启动了。

从 onCreate()函数到 onDestroy()函数运行的时期就是一个 Activity 的完整生命周期。一般情况下。我们会在一个 Activity 的 onCreate()函数中完成各种初始化操作，而在 onDestroy()函数中完成释放内存的操作。然而并不是各个时期 Activity 都是可见的，只有 onResume()函数和 onStop()函数之间的 Activity 是可见的，在 Activity 可见期内，用户可以与 Activity 进行交互，完成所需的功能。

为了帮助读者能够更好地理解，Android 官方提供了一个 Activity 生命周期的示意图，如图 1-1 所示。

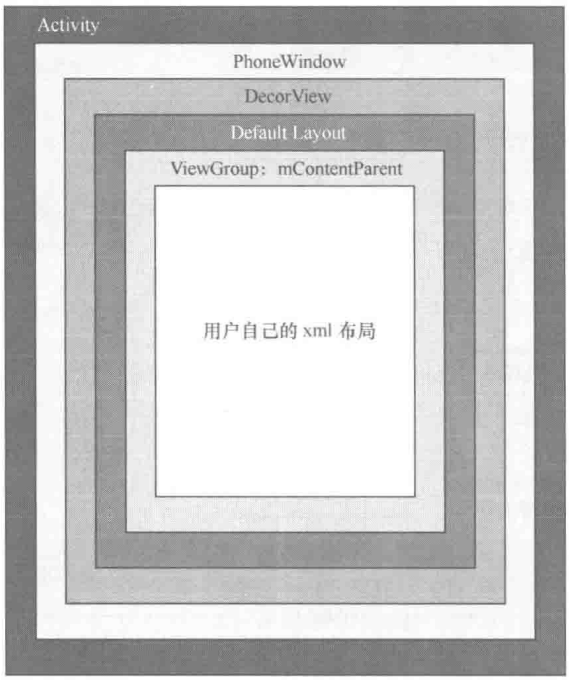


▲图 1-1 Activity 的生命周期

1.1.1 Activity 的构成

Activity 的构成并不是一个 Activity 对象再加上一个布局文件那么简单，在 Activity 和开发人员设置的视图之间还隔着两层。实际上视图会被设置给一个 Window 类，这个 Window 中含有一个 DecorView，这个 DecorView 才是整个窗口的顶级视图。开发人员设置的布局会被设置到这个 DecorView 的 mContentParent 布局中。也就是说 Android 中实际上内置了一些系统布局文件 xml，我们在 xml 中定义的视图最终会被设置到这些系统布局的特定节点之下，这样就形成了整个 DecorView。结构如图 1-2 所示。

从图 1-2 中可以看到，我们的 Activity 之下有一个 PhoneWindow，这个 PhoneWindow 是 Window 的实现类，然后 Window 之下包含一个 DecorView，DecorView 实际上是页面的顶级视图，它从一些系统布局中加载，并且在运行时将开发人员设置给 Activity 的布局资源添加到系统布局的 mContentParent 中。这样一来，用户界面就被添加到系统布局中了，而系统布局会为我们设置好标题栏区域等。



▲图 1-2　Activity 结构

下面就是一个名为 screen_title 的系统布局 xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:fitsSystemWindows="true">
    <!-- Popout bar for action modes -->
    <ViewStub android:id="@+id/action_mode_bar_stub"
                android:inflatedId="@+id/action_mode_bar"
                android:layout="@layout/action_mode_bar"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:theme="?attr/actionBarTheme" />
    <FrameLayout
          android:layout_width="match_parent"
          android:layout_height="?android:attr/windowTitleSize"
          style="?android:attr/windowTitleBackgroundStyle">
      <TextView android:id="@android:id/title"
              style="?android:attr/windowTitleStyle"
              android:background="@null"
              android:fadingEdge="horizontal"
              android:gravity="center_vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent" />
    </FrameLayout>
    <!-这里就是开发人员设置的布局所填充的位置-->
    <FrameLayout android:id="@android:id/content"
          android:layout_width="match_parent"
          android:layout_height="0dip"
          android:layout_weight="1"
          android:foregroundGravity="fill_horizontal|top"
          android:foreground="?android:attr/windowContentOverlay" />
</LinearLayout>
```

上述xml文件中包含了actionbar和标题栏区域，下面就是开发人员设置给Activity的布局区域，这个区域被添加到名为content的布局中，而这整个screen_title.xml又是DecorView的子视图，因此，最终用户界面会显示为标题栏、开发人员设置的界面。例如我们的Activity布局代码如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:gravity="center"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
            android:textSize="30sp"
            android:gravity="center"
            android:layout_height="wrap_content" />

</RelativeLayout>
```

该布局的根视图为RelativeLayout，其中只有一个居中的TextView。运行后的界面如图1-3所示。

jtm_chap01显示的区域就是id为title的TextView，而Hello World就是content布局下的一个子视图。当Activity的onResume函数被调用之后，用户界面就显示在我们面前了。

1.1.2 Activity的4种启动模式

每个应用程序都是由一个或者多个Activity组成，因此，Android内部使用通过回退栈来管理Activity实例。栈是一种后进先出的集合，对于Android来说，当前显示的Activity就在栈顶，当用户点击后退键或者点击应用上的返回按钮，系统就会将栈顶的Activity出栈，此时原来栈顶下的Activity就会变为栈顶显示到设备上。

然而事情可能并不是那么简单，在一些特殊情况下我们可能需要对Activity实例做一些特殊的处理，例如，为了避免重复创建Activity，我们要求一个Activity只有一个实例。好在Android系统为我们提供了这些功能，也就是我们本节要说的Activity的4个启动模式。用户可以在AndroidManifext.xml注册Activity时设置它的启动模式，例如：

```
<activity
    android:name=".MyActivity"
    android:launchMode=" singleTask"
    android:label="@string/app_name" >
</activity>
```

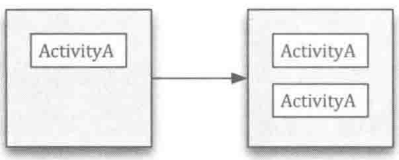
Activity的启动模式有4个，分别为standard、singleTop、singleTask、singleInstance，下面我们逐个介绍它们。

1. standard（标准启动模式）

这是Activity的标准启动模式，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，即在同一个任务栈中可以存在多个 Activity 实例，每个实例都会处理一个 Intent 对象。如果 ActivityA 的启动模式为 standard，并且已经有一个 ActivityA 被启动，在该 ActivityA 中调用 startActivity 时会启动一个新的 ActivityA 实例。栈的变化如图 1-4 所示。

▲图 1-3　用户界面



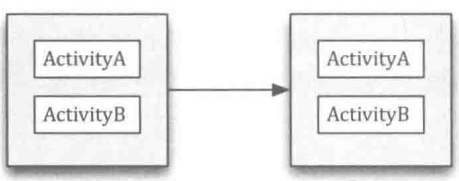
▲图 1-4　栈中有多个 ActivityA 实例

如果 ActivityA 是一个非常耗资源的类，那么将会使它所依附的应用消耗更多的系统资源。

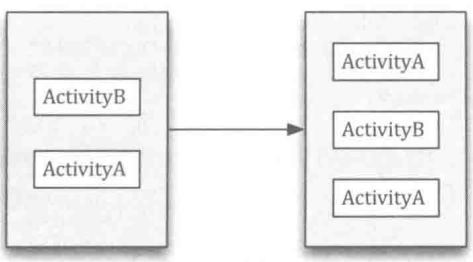
2. singleTop

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() 函数将 Intent 对象传递到这个实例中。例如，ActivityA 的启动模式为 singleTop，并且 ActivityA 的一个实例已经存在于栈顶中。那么再调用 startActivity 启动另一个 ActivityA 时，不会再次创建 ActivityA 的实例，而是重用原来的实例，并且调用原来实例的 onNewIntent()函数。此时任务栈中还是这一个 ActivityA 的实例。栈内变化如图 1-5 所示。

如果以 singleTop 模式启动的 Activity 的一个实例已经存在于任务栈中，但是不在栈顶，那么它的行为和 standard 模式相同也会创建一个新的实例。栈内变化如图 1-6 所示。



▲图 1-5　栈顶的 ActivityA 被重用



▲图 1-6　不在栈顶，重新创建一个 ActivityA