**ABSTRACT**

As an open source mobile operating system, the Android system has occupied about 75% of the market for mobile device operating systems. The development of Android applications takes full advantage of the basic idea of component-based software development, and the entire application is implemented through the combination and communication of components. In order to facilitate the design and development of applications, Android uses Manifest.xml file to configure components and applications, thereby separating the configuration from the complex code logic, which is conducive to the development, design, deployment and operation of the program. Although the Android official document has explained various configuration items in the configuration file, some descriptions do not give precise semantic descriptions. Some instructions are even contradictory to the actual development. Some descriptions do not give the relevant configuration items. Compatible, mutually exclusive relationship. In short, how to accurately configure Android applications is not enough to read official documents. It is necessary to conduct more in-depth research and clarify the exact meaning of these configuration items, so as to provide a basis for better development of programs and reduce design flaws.

This article mainly studies several major aspects of component configuration. First, how to configure the relationship between applications (including components) and processes; second, how to configure the permissions of applications (including components); third, how to configure components correspondence rights constraint relationship. The method adopted is based on reading the official documents, explaining the existence of semantic ambiguity, designing test cases, developing test programs, verifying the accuracy of the relevant configuration items through the program's operating results, and based on this, Summarize several configuration rules for use by designers and developers to improve development efficiency and reduce program security loopholes.

**Key Words:** Android Security;components configuration;permission configuration;

components correspondence

**摘 要**

Android系统作为一种开源的移动终端操作系统，已经占据了移动设备操作系统约75%的市场。Android应用程序的开发充分利用了构件化软件开发的基本思想，通过构件之间的组合和通信实现整个应用程序。为了便于应用程序的设计和开发，Android采用Manifest.xml文件来配置构件和应用程序，从而将配置从复杂的代码逻辑中分离出来，有利于程序的开发、设计、部署和运行。尽管Android官方文档已经就配置文件中的各种配置项进行了说明，但是有些说明没有给出精确的语义描述，有些说明与现实开发甚至是矛盾的，有些说明没有给出相关配置项之间的相容、互斥关系。总之，对于如何准确配置Android应用程序，仅仅通过阅读官方文档是不够的，必须进行更深入的研究，明确这些配置项的确切含义，从而为更好的开发程序、减少设计漏洞提供基础。

本文主要研究组件配置的这样几个主要方面：一是如何配置应用程序（包括组件）与进程之间的关系；二是，如何配置应用程序（包括组件）的权限；三是，如何配置组件间通信的权限约束关系。采用的方法是，在阅读官方文档的基础上，对存在语义模糊的说明，设计测试用例，开发测试程序，通过程序的运行结果，印证或进一步明确相关配置项的准确含义，并在此基础上总结出若干配置规则，以供设计和开发人员使用，提高开发效率，减少程序安全漏洞。

**关键词：**Android Security；组件配置；权限配置；组件通信

**目 录**

**[1 绪论](#_Toc516593041)** [1](#_Toc516593041)

[1.1 研究背景及意义 1](#_Toc516593042)

[1.2 国内外研究现状及进展 2](#_Toc516593043)

[1.3 论文结构 3](#_Toc516593044)

**[2 Android组件配置及安全机制](#_Toc516593045)** [4](#_Toc516593045)

[2.1 Android四大组件及配置 4](#_Toc516593046)

[2.1.1 四大组件概述 4](#_Toc516593047)

[2.1.2 AndroidManifest.xml配置文件详解 5](#_Toc516593048)

[2.2 Android安全机制概述 6](#_Toc516593049)

[2.2.1 基于Linux文件系统权限特性 6](#_Toc516593050)

[2.2.2 Android危险权限和权限组 6](#_Toc516593051)

[2.3 packages.xml文件概述 7](#_Toc516593052)

[2.4 ADB工具的基本使用方法 8](#_Toc516593053)

**[3 应用程序的基本属性配置](#_Toc516593054)** [10](#_Toc516593054)

[3.1 应用程序共享用户ID 10](#_Toc516593055)

[3.1.1 sharedUserId属性值的正确格式 10](#_Toc516593056)

[3.1.2 多个APP共享用户ID 11](#_Toc516593057)

[3.1.3 sharedUserId值为另一个已安装的包名 11](#_Toc516593058)

[3.2 应用程序包的覆盖 12](#_Toc516593059)

[3.3 定义但不注册组件 12](#_Toc516593060)

[3.4 含有多个MainActivity 13](#_Toc516593061)

**[4 应用程序及组件进程分析](#_Toc516593062)** [15](#_Toc516593062)

[4.1 process的正确格式 15](#_Toc516593063)

[4.2 应用程序及其组件的优先级 16](#_Toc516593064)

[4.3 将component部署到其它进程中 16](#_Toc516593065)

[4.4 exported属性的含义及其缺省值 17](#_Toc516593066)

[4.5 multiprocess属性与组件实例化分析 19](#_Toc516593067)

[4.5.1 启动和被启动的activity在同一App中 19](#_Toc516593068)

[4.5.2 启动和被启动的activity不在同一App中（签名不同） 21](#_Toc516593069)

[4.5.3 启动和被启动的activity不在同一App中（签名相同） 22](#_Toc516593070)

**[5 应用程序和组件权限配置研究](#_Toc516593071)** [25](#_Toc516593071)

[5.1 共享用户间APP的权限集合 25](#_Toc516593072)

[5.2 enforced权限的继承 26](#_Toc516593073)

[5.3 本章小结 29](#_Toc516593074)

**[6 组件间通信的权限分析](#_Toc516593075)** [30](#_Toc516593075)

[6.1 Intent的隐式启动 30](#_Toc516593076)

[6.2由Intent action所施加的权限 30](#_Toc516593077)

[6.3 本章小节 31](#_Toc516593078)

**[7 总结](#_Toc516593079)** [32](#_Toc516593079)

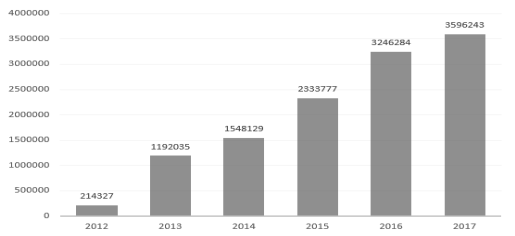
**[致谢](#_Toc516593080)** [33](#_Toc516593080)

**[参考文献](#_Toc516593081)** [34](#_Toc516593081)

**1 绪 论**

**1.1 研究背景及意义**

随着科学技术的迅速发展，智能移动设备因其功能丰富、便于携带、易操作等多种优点，在人们日常生活中占据着越来越重要的地位。不同的移动设备搭载着不同的操作系统，最常见的操作系统是Android。统计数据显示，2017年，销售的所有智能手机中，Android系统的智能手机更以85.9%的市场份额领跑全球。Android是一个基于Linux内核的开放源代码移动操作系统，由Google成立的Open Handset Alliance（OHA，开放手持设备联盟）持续领导与开发[1]。从2007年11月5日发行至今，Android系统在不断完善，综合了其他操作系统的许多优势及优点，同时在设计时也充分考虑了安全性，但是这并不意味着Android系统绝对的安全可靠[2]，总有不法分子利用Android系统自身的漏洞，实施信息攻击，盗取用户信息等。去年1月份，德国网络安全公司GDATA官网的报告显示，恶意软件的数量在逐年递增，仅在2017 年第一季度就有750,000个新恶意软件文件，如图所示。可见，Android安全问题是目前最需要解决的问题之一。



**图1-1 2012~2017年Android恶意软件数量**

为了保证Android应用程序的安全性，必须充分利用 Android 系统提供的权限机制。使用权限可以保护敏感资源不被非法访问，也可以保护组件间交互的安全性。组件交互的安全性主要通过对应用中每个组件的权限配置来实现。Android四大基本组件分别是Activity活动，Service服务,Content Provider 内容提供者，BroadcastReceiver广播接收器，在这些组件通信过程中，可能由于开发者对其权限配置不当，产生一些安全漏洞。虽然Android官方文档对组件配置进行了说明，但是对于这些配置的含义并没有给出严格定义，对于不同组件配置之间的关系也没有给出深入细致的描述，使得应用开发缺少可依赖的语义基础[3]。本论文试图通过设计一系列测试案例，对组件间的配置进行测试，通过观察测试结果，从而确认组件配置的准确含义，为Android应用开发者提供语义基础。

**1.2 国内外研究现状及进展**

Android 应用的安全问题日益显著， Android 系统存在的风险主要是：恶意扣除手机话费，泄露用户个人信息，传播此类恶意软件等。我们应该如何应对Android应用中存在的安全隐患，从而对移动设备进行防护。现在有国内外的众多学者为此做了许多研究。

William Enck、Machigar Ongtang、Patrick McDaniel 3人发表文章[4]，通过一系列案例向我们描述了Android的安全模型。为了进一步检测Android的安全策略，他们使用Kirin工具从清单文件中提取应用程序的安全策略，以确定请求的权限和组件分配的权限是否一致，通过使用运行在Android上的Prolog引擎生成符合性证明。如果应用程序的策略不符合要求，则不会安装应用程序。他们还设计并实现了一个基于权限的恶意程序分析框架用于检测多种敏感权限组合类的程序[5]。

FANG等人[6]从粗粒度权限管理、不完整的权限文档、权限分配过高等6个方面对Android安全展开分析，认为Android安全是建立在我们开发过程中权限控制上面[7]，尤其是较多的安全性相关的数据公开可用，权限配置显得更加重要。

Michael Grace等[8]设计了一个工具Woodpecker[9]，它可以检测Android的权限是否发生泄漏。Woodpecker采用静态分析技术，对程序构建控制流图，标记处带权限的操作，跟踪这些语句，观察语句是否被第三方应用程序调用，以此有效的检测显示和隐式的应用程序的权限是否发生泄漏。

William Enck、Asaf Shabtai等人[10]研究了Android系统平台上常用的多种安全防范措施，为了进一步了解智能手机的安全性，他们对上千种流行的免费的Android应用程序的展开了一系列研究。使用deb反编译器直接对App进行反编译，并对产生的2100行代码进行静态分析，对其中的危险功能和漏洞进行了广泛的研究，大大的提升了Android系统的整体架构的安全性。

国内Hui Ye等人[11]提出提出了一种自动化测试方案---DroidFuzzer，该方案可以对接受外部MIME数据的Activity组件展开测试。

Chan等人通过污点检测的静态分析方法对权限泄露问题进行了检查，设计并实现了 DroidChecker检测是否发生显示权限的泄露。另一方面，应用程序中的权限申请需要满足最小原则，只有在程序中必需的权限才予以分配，这样才能最大化的发挥Android权限机制，这样即便应用程序因为漏洞而被攻击者攻击，也能够将程序的被利用程度控制在一个小的权限范围内。

**1.3 论文结构**

本文的主要内容是熟悉组件配置特性，如组件与进程之间的关系、共享用户ID等，并通过设计一系列实验，验证组件配置及安全特性的准确含义，为Android应用开发提供语义基础。本文共有七章，结构如下：

第一章，绪论。本章介绍研究背景、意义以及国内外现状，主要描述了当前阶段Android市场的发展状况和面临的安全问题，要减少恶意APK对Android用户造成的威胁必不可少的步骤就是在开发过程中降低隐患。

第二章，Android组件配置及安全机制。简单介绍了Android系统的架构及Android的四大组件的基本概念，详细讲解了AndroidManifest.xml文件和packages.xml文件的作用及其属性，还介绍了Android的安全机制和后续调试过程中会用到的ADB工具的常用命令。

第三章，应用程序（Application）的基本属性配置。通过设计测试用例，对应用程序共享用户ID的条件进行了分析，通过实验验证程序包覆盖安装的条件，分析了存在多个MainActivity的情况。

第四章，应用程序及组件进程配置分析。本章主要是对应用程序（包括组件）与它们所运行的进程之间的配置关系展开了一系列测试，包括单个App进程的部署、多个App间的跨应用访问，multiprocess对组件实例可能产生的影响等。

第五章，应用程序和组件权限配置研究。通过多个案例的结果分析共享用户间App的权限集合和enforced权限的继承。

第六章，组件间通信的权限分析。分析了intent的隐式启动对App本身可能产生的隐患以及intent action的权限约束。

第七章，总结。本章对毕设和论文的工作内容进行了总结。

**2 Android组件配置及安全机制**

Android系统为了达到结构清晰，层次更加分明，使用了分层的架构。Android系统一般分为4层，从高层向低层分别是应用程序层、应用程序框架层、系统运行库层和Linux核心层，层与层之间的作用各不相同，但是相互关联，协同工作，整体架构如图2-1：



**图2-1 Android体系结构**

**2.1 Android四大组件及配置**

**2.1.1 四大组件概述**

Android四大组件：Activity、Service、Broadcast Receiver、Content Provider。

Activity是Android程序与用户交互的窗口，从视觉效果来看，一个Activity占据当前的窗口，响应所有窗口事件，具备有控件，菜单等界面元素。从内部逻辑来看，Activity需要为了保持各个界面状态，需要做很多持久化的事情，还需要妥善管理生命周期，和一些转跳逻辑。

Service就是剥离了界面的Activity，它们在很多Android的概念方面比较接近，都是封装一个完整的功能逻辑，通常都是后台长时间运行，接受上层指令，完成相关事务的模块。

Broadcast Receiver 接收一种或者多种Intent跳转做触发事件，接受相关消息，做一些简单的处理，转换成一条Notification，统一了Android的事件广播模式。

Content Provider 提供第三方应用数据的访问方案。可以派生ContentProvider类，对外提供数据，像数据库一样进行选择排序，屏蔽内部数据的存储细节，向外提供统一的接口模型，大大简化了上层应用，对数据的整合提供了更方便的途径[12]。

**2.1.2 AndroidManifest.xml配置文件详解**

AndroidManifest.xml是Android应用程序中最重要的文件之一。该文件是Android应用程序中权限声明和定义配置的文件，每个应用的根目录中都必须包含一个AndroidManifest.xml文件，描述了包中的组件、全局数据和各自的实现类等，如表2-1所示是一些经常使用到的元素：

**表2-1 AndroidManifest.xml文件中常见元素及其描述**

|  |  |
| --- | --- |
| **元素** | **描述** |
| <manifest> | 根节点，对包中所包含的内容进行描述 |
| <uses-permission /> | 应用程序正常运行申请的权限，可以申请多个权限，即一个.xml文件中可以有多个此元素 |
| <permission /> | 声明一个安全授权，限定其他应用程序能否访问此程序中的资源，一个xml文件可以包含多个此元素 |
| <instrumentation /> | 声明用于测试组件的代码，一个xml文件可以包含多个此元素 |
| <uses-sdk /> | 用于声明当前应用程序能够兼容的最低版本号的SDK |
| <application> | 包中应用程序组件声明的根节点 |
| <activity> | 用户与应用程序交互的桥梁，当用户点击应用时，出现在手机上的主页面就是一个activity |
| <intent-filter> | 声明组件支持的Intent值 |
| <action /> | 声明组件支持的Intent Action |
| <category /> | 声明组件支持的Intent Category |
| <data /> | 声明组件支持的数据类型或URI |
| <meta-data /> | 一个key-value，是父组件添加的数据，可以是activity等 |
| <service> | 运行在后台且可以运行任意时间 |
| <receiver> | Intent Receiver对应用程序读取的数据进行修改 |
| <provider> | 管理应用程序中所涉及的数据，并将数据传递给需要的组件 |



<manifest>和<application>元素是必需的，它们都必须存在并且只能出现一次。其他大部分元素可以出现多次或者根本不出现。

同一级别的元素通常不分先后顺序。例如，<activity>、<provider>和<service>元素可以按任何顺序混合在一起。这条规则有两个主要例外：<activity-alias>元素必须跟在别名所指的<activity>之后。<application>元素必须是<manifest>元素内最后一个元素。换言之，</manifest>结束标记必须紧接在</application>结束标记后。

根据Android开发者文档可知，如果一个组件定义了intent-filter（intent过滤器）子元素，则默认表示该组件是被暴露在Android系统环境中，可以被接收来自其他应用程序的Intent。若组件定义了intent-filter标签但是不想该组件参与跨应用调用，则可以为该的组件添加android:exported="false"属性来说明该组件是私有的，只有同一个应用程序的组件或带有相同用户ID的应用程序才能启动该组件或绑定该服务。

当我们要创建应用时，需要通过AndroidManifest.xml文件声明应用程序的访问权限。如果应用需要访问受权限保护的功能，则必须在清单中使用<uses-permission>元素声明应用需要该权限（注：新权限用<permission>元素来声明），如果授予权限，则应用能够使用受保护的功能[13]。否则，其访问这些功能的尝试将会失败，并且不会向用户发送任何通知。如果组件定义了android:permission属性，则表示该组件在被调用的时候需要申请对应的权限，如果该权限的级别是signature或signatureOrSystem则该组件只能被授权被具有相同签名的应用程序，这样该组件即便是暴露在外，也因为其他程序无法获得调用权限而不会受到攻击[5][6]。在申请权限时候我们也要注意，尽可能遵循“最小权限原则”，避免申请的权限过多，因为这会增加遭受攻击的风险[7][8]。

**2.2 Android安全机制概述**

**2.2.1 基于Linux文件系统权限特性**

Android是一个权限分离的系统。这是利用Linux已有的权限管理机制，通过为每一个Application分配不同的uid和gid，从而使得不同的Application之间的私有数据和访问达到隔离的目的。与此同时，Android还在此基础上进行扩展，提供了permission机制，它主要是用来对Application 可以执行的某些具体操作进行权限细分和访问控制[14]。

**2.2.2 Android危险权限和权限组**

危险权限涵盖应用需要涉及用户隐私信息的数据或资源，或者可能对用户存储的数据或其他应用的操作产生影响的区域。例如，能够读取用户的联系人属于危险权限。如果应用声明其需要危险权限，则用户必须明确向应用授予该权限。危险权限如下表：

**表2-2 Android危险权限组名和权限名**

|  |  |
| --- | --- |
| **权限组名** | **权限名** |
| CALENDAR | READ\_CALENDAR |
| WRITE\_CALENDAR |
| CAMERA | CAMERA |



|  |  |
| --- | --- |
| CONTACTS | READ\_CONTACTS |
| READ\_CONTACTS |
| GET\_CONTACTS |
| LOCATION | ACCESS\_FINE\_LOCATION |
| ACCESS\_COARSE\_LOCATION |
| MICROPHONE | RECORD\_AUDIO |
| PHONE | READ\_PHONE\_STATE |
| CALL\_PHONE |
| READ\_CALL\_LOG |
| WRITE\_CALL\_LOG |
| ADD\_VOICEMAIL |
| USE\_SIP |
| PROCESS\_OUTGOING\_CALLS |
| SENSORS | BODY\_SENSORS |
| SMS | SEND\_SMS |
| RECEIVE\_SMS |
| READ\_SMS |
| RECEIVE\_WAP\_SMS |
| RECEIVE\_MMS |
| STORAGE | READ\_EXTERNAL\_STORAGE |
| WRITE\_EXTERNAL\_STORAGE |

如果应用请求其清单中列出的危险权限，而应用目前在权限组中没有任何权限，则系统会向用户显示一个对话框，描述应用要访问的权限组[16]。

**2.3 packages.xml文件概述**

Android系统在初始化时，PackageManager的底层实现类PackageManagerService回去扫描系统的/data/system/目录下的packages.xml文件，这个文件包含了系统当中安装的APK的所有属性，权限等信息，当APK安装、删除升级时，这个文件同样也会更新。

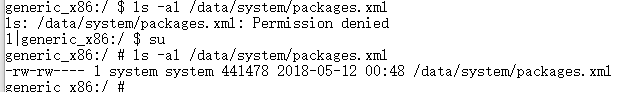
<permissions>标签定义了目前系统中定义的所有权限。主要分为两类：系统定义的（package属性为Android）和APK定义的（package属性为APK的包名）。

<package>代表一个APK的属性。属性含义如表2-3所示：

**表2-3 package属性主要元素及其含义**

|  |  |
| --- | --- |
| **元素** | **含义** |
| name | APK的包名 |
| codePath | 安装路径。有/system/app系统APK和/data/app两种。/system/app存放系统出厂时预置的一些APK，/data/app存放用户安装的第三方APK |
| system | 如果APK被安装在/system/app下，system的值为true；安装在/data/app下面的话，值为true |
| ts | 时间戳 |
| version | APK的版本号 |
| userId | Android系统启动一个普通的APK时，会为这个APK分配一个独立的UID，这就是userId。如果APK要和系统中其它APK使用相同的UID的话，那就是sharedUserId |
| perms | 对应清单文件的<uses-permission>标签，记录Apk权限信息 |

访问packages.xml文件需要系统有root权限，使用命令“su”切换为root，如图2-2所示方法：



**图2-2 packages.xml访问方法**

**2.4 ADB工具的基本使用方法**

adb.exe工具（Android Debug Bridge，ADB），存在于SDK的platform-tools目录中，允许开发人员与模拟器或者连接的Android设备进行通信。

ADB的常见指令介绍如表2-4所示：

**表2-4 ADB常用命令**

|  |  |
| --- | --- |
| **命令** | **描述** |
| adb start-server | 开启adb服务 |
| adb devices | 列出所有设备 |
| adb logcat | 查看日志 |
| adb kill-server | 关闭adb服务 |
| adb shell | 挂载到Linux的空间 |

**续表2-4 ADB常用命令**

|  |  |
| --- | --- |
| adb -s<模拟器名称>install<应用程序（加扩展名）> | 安装应用程序到指定模拟器 |
| adb uninstall <package> | 卸载APK |
| adb reboot | 重启设备 |
| adb shell kill [pid] | 杀死一个进程 |
| adb shell ps -x [PID] | 查看指定进程状态 |
| adb shell cat /proc/[pid]/status | 查看进程pid的状态 |

**3 应用程序的基本属性配置**

**3.1 应用程序共享用户ID**

在APP成功安装后，Android系统会为其自动分配一个userId和userName，在设计和安装时，用户是无法指定和预测这两个属性值的。两个或多个APPs可以由设计者指定为共享用户ID，即在每个APP的Manifest.xml文件中，将<manifest>元素的android:sharedUserId属性指定为同一用户ID。本节主要测试共享用户ID时应满足的条件。

在应用程序的基本属性配置中，最复杂的是共享用户ID的应用程序配置。本节通过设计测试用例，枚举不同的配置条件，观察配置结果，从而得到配置共享用户ID的应用程序的条件。测试用例的设计主要拟覆盖如下配置关系：

（1）sharedUserId属性值的正确格式；

（2）配置共享用户ID的应用程序的条件；

（3）应用程序包更新的条件。

**3.1.1 sharedUserId属性值的正确格式**

根据Android官方文档，配置共享用户ID的应用程序时，需要设置AndroidManifest.xml文件中<application>标签的shardUserId属性的值，但是对于该属性值应当遵守格式没有给出严格说明。设计本实验的目的就是明确shardUserId属性的值格式。shardUserId属性的值为字符串类型，根据开发实践中的常见情况，我们将可能的字符串取值分为三类：

① android:sharedUserId=” ”，即取值为空字符串；

② android:sharedUserId取值为Java包格式，即xxx.xxx.xxxx形式；

③ android:sharedUserId取值为非Java包格式。

针对这三种情况，设计了三个测试用例，分别观察该设置对应用程序的影响。这种影响又可进一步分为两类：

① 是否能够成功安装；

② 在能够正常安装的前提下，是否能够正常运行。

由于这些测试用例的设计比较直观，这里我们直接给出测试结果，如表3-1所示：

**表3-1 shareUserId的取值及其测试结果**

|  |  |  |
| --- | --- | --- |
| **sharedUserId的取值** | **测试结果** | **说明** |
| 空字符串 | 输出值为null，可以正常运行 | 不对空字符串解析 |
| java包格式 | 输出值为”com.demo.test”，可以正常运行 | 支持Java包格式的字符串 |
| 非Java包格式 | 安装失败 | 不支持非Java包字符串 |

从该组测试用例中，可以总结如下规则：

规则1：设置应用程序的sharedUserID属性值时，必须遵守Java包格式，否则出现安装错误。

规则2：设置应用程序的sharedUserID属性值时，如果值为空字符串，等价于没有设置sharedUserID，不对空字符串解析。



**3.1.2 多个APP共享用户ID**

从3.1.1的案例中知道sharedUserID属性值必须遵守Java包格式。如果是多个APP设置相同的sharedUserID，官方文档中并没有给出详细的说明多个APP间需要满足什么条件。本案例将对其需要满足的条件进行研究，对常见情况分为两种：

①多个APP的签名不相同；

②多个APP的签名相同。

针对以上两种情况，设计两个测试案例，设APP1和APP2是两个拟共享用户ID的应用程序。APP1.android:sharedUserId和APP2.android:sharedUserId的值都采用正确格式且取值相同。测试结果如表3-2所示：

**表3-2 多个APP共享用户ID测试及结果**

|  |  |  |  |
| --- | --- | --- | --- |
| **签名** | **APP编号** | **测试结果** | **说明** |
| 签名不相同 | APP1 | 安装成功并运行 | 安装第2个APP会出错，两个APP只能同时安装1个 |
| APP2 | 安装失败 |
| 签名相同 | APP1 | 安装成功并运行 | 两个APP可以同时安装并且userID相同 |
| APP2 | 安装成功并运行 |

从该组测试用例中，可以总结如下规则：

规则1：设置多个应用程序的sharedUserID属性为相同值时，即共享用户ID的APP间，所有的APP签名必须相同。

规则2：如果两个或两个以上APP的签名相同，则他们的userID相同，即共享用户ID。

规则3：签名相同与sharedUserID相同互为充要条件。



**3.1.3 sharedUserId值为另一个已安装的包名**

APP1的包名为”com.demo.app1”，但未设置sharedUserID值，安装APP1；修改APP2的sharedUserID值为”com.demo1.app1”，安装APP2。测试结果如表3-3所示：

**表3-3 sharedUserID为已安装的包名测试结果**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **APP编号** | **sharedUserID实际值** | **UserName** | **ProcessName** | **说明** |
| APP1 | NULL | u0\_a129 | com.demo1.app1 | 有两个进程项，而且用户名不同 |
| APP2 | com.demo1.app1 | u0\_a130 | com.demo1.app2 |

从该测试用例中，可以总结如下规则：

规则1：不可以为了简便直接将某个APP的sharedUserID设置为某个已经安装的包名，这样是不可以实现用户共享的。



**3.2 应用程序包的覆盖**

当拟安装的APP的包名与Android系统中的包不同名时，认为这是一个新包；当拟安装的APP与系统中某个包名重复时，被认为是一个更新包。但是文档中没有说明除了包名相同，还应该满足什么条件。本案例测试更新包应满足的条件，分为两种情况：

①两个APP签名不相同；

②两个APP签名相同。

设计两个测试案例，APP1的包名为”com.demo.app1”，将App2的包名也修改为”com.demo.app1”。

**表3-4 覆盖安装条件的测试及结果**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **签名** | **APP编号** | **测试结果** | **UserID** | **说明** |
| 签名不相同 | APP1 | 安装成功并运行 | u0\_a138 | 安装第2个APP会出错，需要卸载第1个后方可安装第2个 |
| APP2 | 安装失败，卸载第1个后方可安装 | u0\_a140 |
| 签名相同 | APP1 | 安装成功并运行 | u0\_a143 | APP2对APP1进行了覆盖 |
| APP2 | 安装成功并运行 | u0\_a143，安装后，只有1个进程项 |

从该组测试用例中，可以总结如下规则：

规则1：包名相同，签名不同时，后续APP在安装中会出错；当包名、签名均相同的时候，后续APP在安装过程中会覆盖先前的APP。



**3.3 定义但不注册组件**

Android系统在实例化一个Activity时，需要查看xml文件的配置，如果没有注册，是不会启动的。现在Java文件中定义了一个Activity类，但不在AndroidManifest.xml中注册这个类，那么当用类名启动该activity时，能否成功？

在项目APP2中新建一个名为SecondActivity，在FirstActivity中添加一个button，当点击button时启动SecondActivity。在activity\_first\_layout中添加button：

<Button

android:id="@+id/SecondActivity"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="SecondActivity "

android:textAllCaps="false"/>

在FirstActivity中为button的点击事件注册一个监听器：

Button button = (Button)findViewById(R.id.SecondActivity);

button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

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

startActivity(intent);

}

});

去掉Androidmanifest.xml中对secondActivity的注册：

<activity

android:name="com.example.second.SecondActivity">

</activity>

安装运行APP2，点击button试图启动secondActivity，程序崩溃，提示“APP2 has stopped”。

从该测试用例中，可以总结如下规则：

规则1：定义一个组件（component）后，必须在AndroidManifest.xml中注册，否则不可以使用。



**3.4 含有多个MainActivity**

从官方文档中，了解到一个APP不可以没有MainActivity，那么如果含有多个MainActivity呢？本案例用于测试含有多个MainActivity时程序是否可以正常运行。将情况分为两种：

①FirstActivity注册位置在前，SecondActivity注册位置在后；

②SecondActivity注册位置在前，FirstActivity注册位置在后；

代码片段如图3-1所示：



**图3-1 FirstActivity和SecondActivity注册位置代码片段**

MainActivity注册位置不同，产生的测试结果如表3-5所示：

**表3-5 多个MainActivity测试结果**

|  |  |  |  |
| --- | --- | --- | --- |
| **Activity** | **注册位置** | **进程名** | **说明** |
| FirstActivity | 在前 | com.test.multimain1 | 可以含有多个MainActivity |
| SecondActivity | 在前 | com.test.multimain2 |

从该测试用例中，可以总结如下规则：

规则1：可以定义多个MainActivity，但是以第一个“MainActivity”作为主活动。



**4 应用程序及组件进程分析**

可以使用process属性来指定应用程序或者其组件的进程，如果不设置process属性，那么activity启动后，默认运行在其application的进程中。但是如果设置了process属性，那么在一定条件下，可以把activity组件部署到其他进程中。另外，application和activity都有process属性，因此可能同时设置了process属性，这时该按谁的设置来部署呢？也就是哪一个的process属性优先级更高？这就是本小节的目的。

**4.1 process的正确格式**

根据Android官方文档，可以设置AndroidManifest.xml文件中<application>标签或component的process属性的值，但是对于该属性值应当遵守格式没有给出严格说明。设计本实验的目的就是明确process属性的值格式。process属性的值为字符串类型，根据开发实践中的常见情况，我们将可能的字符串取值分为四类：

①不设置process属性，process值缺省；

②process值为非Java包格式；

③process值为Java包格式；

④process职位”:remote”格式。

针对这四种情况，设计了四个测试用例，分别观察设置对应用程序的影响。这种影响又可进一步分为两类：

① 是否能够成功安装；

② 在能够正常安装的前提下，是否能够正常运行。

由于这些测试用例的设计比较直观，这里我们直接给出测试结果，如表4-1所示：

**表4-1 process取值及测试结果**

|  |  |  |  |
| --- | --- | --- | --- |
| **process取值** | **是否可以**  **正常运行** | **进程名** | **说明** |
| 未设置 | 是 | com.demo.app1 | 默认为包名 |
| 非Java包格式 | 否，安装出错 | / | 不支持非Java包格式 |
| Java包格式 | 是 | a.b.c.d.e.f | 支持Java包格式 |
| :remote格式 | 是 | com.demo.app1:remote | 包名+“:remote” |

从该测试用例中，可以总结如下规则：

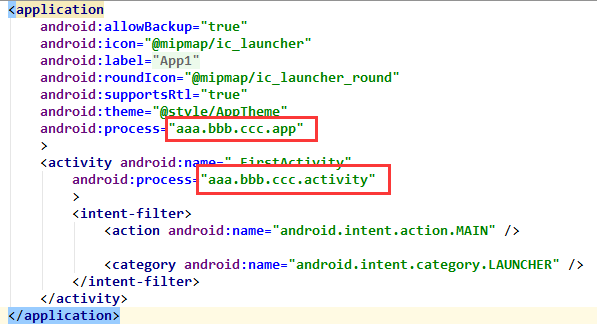
规则1：设置应用程序的process属性值时，必须遵守Java包格式，否则安装会出现错误。

规则2：未设置process属性时，应用程序（component）进程名默认为项目包名。



**4.2 应用程序及其组件的优先级**

根据文档可以知道，application和activity标签下均可以设置process属性，但是没有说他们两个哪个的优先级更高，本案例用于测试application及其component的优先级。Process属性设置如图4-1所示：



**图4-1 process属性设置**

application.process设置为“aaa.bbb.ccc.app”，application.activity.process设置为“aaa.bbb.ccc.activity”，当我们运行APP后，可以看到进程名为“aaa.bbb.ccc.activity”。

从该测试用例中，可以总结如下规则：

规则1：当同时在application和component设置process属性时，component优先级更高。



**4.3 将component部署到其它进程中**

从前边的测试可知，默认情况下，一个APP启动后，将会运行在默认进程中，进程名为包名，但是当设置了<application>标签下的android:process属性后，该APP将被部署到另一个进程中。本案例用于测试将进程部署到其它进程需要满足的条件，分为以下2种情况：

①共享用户ID；

②非共享用户ID。

根据以上情况，设计2个测试案例，结果如表4-1所示：

**表4-1 将组建部署到其他进程测试结果**

|  |  |  |  |
| --- | --- | --- | --- |
| **是否共享用户ID** | **APP安装情况** | **进程数目** | **说明** |
| 共享用户ID | 先安装APP1，再安装APP2 | 1 | 只有1个进程项，部署成功 |
| 非共享用户ID | 2 | 2个进程项，只是进程名相同 |

从该测试用例中，可以总结如下规则：

规则1：同一用户下的component可以通过设置process共享地址空间，而当非同一用户时，即使设置process相同，它们只是有着相同的进程名，并没有共享地址空间；

规则2：两个不同APK中的组件运行于同一个进程，需要以下三个条件：两个应用程序使用相同的android:sharedUserId，两个应用使用相同的keystore进行签名，为组件设置相同的android:process。



**4.4 exported属性的含义及其缺省值**

Activity的exported属性表示该组件是否向外公开，即是否可以被其它应用所启动。

现在有App1和App2两个项目，app1和app2的签名不同，shareduserID也不相同。

App1有只有一个名为“FirstActivity”的主活动，FirstActivity包含3个button；项目app2有3个activity，分别是：FirstActivity、SecondActivity和ThirdActivity，FirstActivity含有2个button。

App1中FirstActivity的3个button分别用于启动app2的3个activity。App2中FirstActivity中的2个button分别用于启动此app中的另外两个activity。

项目app1的activity\_first\_layout.xml添加：

<Button

android:id="@+id/App2FirstActivity"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text=" App2:FirstActivity"

android:textAllCaps="false"/>

<Button

android:id="@+id/App2SecondActivity"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text=" App2:SecondActivity"

android:textAllCaps="false"/>

<Button

android:id="@+id/App2ThirdActivity"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text=" App2:ThirdActivity"

android:textAllCaps="false"/>

在项目app1的FirstActivity中onCreate函数添加button的监听器：

Button button\_app2\_first\_activity = (Button)findViewById(R.id.App2FirstActivity);

button\_app2\_first\_activity.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent();

intent.setClassName("com.demo.app2",

"com.demo.app2.FirstActivity");

startActivity(intent);

}

});

Button button\_app2\_second\_activity = (Button)findViewById(R.id.App2SecondActivity);

button\_app2\_second\_activity.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent();

intent.setClassName("com.demo.app2",

"com.demo.app2.SecondActivity");

startActivity(intent);

}

});

Button button\_app2\_third\_activity = (Button)findViewById(R.id.App2ThirdActivity);

button\_app2\_third\_activity.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent();

intent.setClassName("com.demo.app2",

"com.demo.app2.ThirdActivity");

startActivity(intent);

}

});

项目app2的activity\_first\_layout.xml添加：

<Button

android:id="@+id/SecondActivity"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="SecondActivity"

android:textAllCaps="false"/>

<Button

android:id="@+id/ThirdActivity"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="ThirdActivity"

android:textAllCaps="false"/>

在项目app2的FirstActivity中onCreate函数添加button的监听器：

Button button\_second\_activity = (Button)findViewById(R.id.SecondActivity);

button\_second\_activity.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

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

startActivity(intent);

}

});

Button button\_third\_activity = (Button)findViewById(R.id.ThirdActivity);

button\_third\_activity.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

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

startActivity(intent);

}

});

现在使用APP1启动APP2中的activity，运行APP1，点击button“App2:Seco ndActivity”：启动失败，APP1崩溃。提示“APP1 has stopped”。

此时在项目APP2的SecondActivity添加“exported=true”，再次使用App1启动App2的activity，可以启动成功。

从该测试用例中，可以总结如下规则：

规则1：exported默认是false，即该组件不可以被外部访问。



**4.5 multiprocess属性与组件实例化分析**

只有provider和activity定义了android:multiprocess。Activity的multiprocess属性表明activity能否在启动它的进程中被实例化。当multiprocess=false时，该Activity的实例只能在指定的进程中实例化；当multiprocess=true时，哪个组件启动它，它就在那个组件所在的进程中实例化。显然multiprocess的语义与process属性以及launchmode属性有密切关系。这小节测试multiprocess的语义，以及multiprocess与process属性设定值之间的关系，也就是当二者冲突时，哪一个属性的优先级更高。

**4.5.1 启动和被启动的activity在同一App中**

现在有项目App2，它有3个activity：FirstActivity、SecondActivity、ThirdActivity，其中FirstActivity是MainActivity。在每个activity中都可以启动另外两个activity。如下：

#### 4.5.1.1 activity.process未设置

|  |
| --- |
|  |

（1）multiprocess=false时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即是否总在默认的app进程中？

① MainActivity->SecondActivity

MainActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2

②MainActivity->ThirdActivity->SecondActivity

MainActivity进程名：com.demo.app2

ThirdActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2

（2）multiprocess=true时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？

①MainActivity->ThirdActivity

MainActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2

②MainActivity-> SecondActivity->ThirdActivity

MainActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2

ThirdActivity进程名：com.demo.app2

#### 4.5.1.2 activity.process有值

将activity.process设置为“:remote”。

（1）multiprocess=false时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即是否在“:remote”进程中？

①MainActivity->SecondActivity

MainActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2:remote2

② MainActivity->ThirdActivity->SecondActivity

MainActivity进程名：com.demo.app2

ThirdActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2:remote2

（2）multiprocess=true时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？

①MainActivity->ThirdActivity

MainActivity进程名：com.demo.app2

ThirdActivity进程名：com.demo.app2

② MainActivity-> SecondActivity->ThirdActivity

MainActivity进程名：com.demo.app2

SecondActivity进程名：com.demo.app2:remote2

ThirdActivity进程名：com.demo.app2

#### 4.5.1.3 本节小结

同一App中多个component：①当process未设置时，无论multiprocess为何值，组件被启动时与启动组件的进程无关，总是运行在默认app进程中；②当process设置相同时，multiprocess为“false”时，组件所在的进程与启动它的组件所在进程无关；反之，multiporcess为“true”时，组件所在的进程与启动它的组件所在进程有关。

**4.5.2 启动和被启动的activity不在同一App中（签名不同）**

在app2的activity标签下添加”exported=true”。

app1有3个activity：FirstActivity、SecondActivity、ThirdActivity，其中FirstActivity为MainActivity，app1中的每个activity都可以启动app2中的任意一个activity。将app1的SecondActivity进程设置为“:second”，将app1的ThirdActivity进程设置为“:third”。具体操作与4.4一样。

#### 4.5.2.1 activity.process未设置

（1）multiprocess=false时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即是否总在默认的app进程中？

①App1.MainActivity->App1.SecondActivity-> App2.SecondActivity

App1.SecondActivity进程名：com.demo.app1:second

App2.SecondActivity进程名：com.demo.app2

②App1.MainActivity-> App1.ThirdActivity-> App2.SecondActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2.SecondActivity进程名：com.demo.app2

（2）multiprocess=true时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即被启动的activity能够在另一个app中被实例化？

①App1.MainActivity->App1.SecondActivity-> App2.ThirdActivity

App1.SecondActivity进程名：com.demo.app1:second

App2. ThirdActivity进程名：com.demo.app2

②App1.MainActivity-> App1.ThirdActivity-> App2. ThirdActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2. ThirdActivity进程名：com.demo.app2

#### 4.5.2.2 activity.process有值

activity.process设置为“:remote”。

<activity android:name=".SecondActivity"

android:label="App2-SecondActivity"

android:multiprocess="false"

android:process=":remote2">

</activity>

<activity android:name=".ThirdActivity"

android:label="App2-ThirdActivity"

android:multiprocess="true"

android:process=":remote3">

</activity>

（1）multiprocess=false时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即是否总在“:remote”进程中？

①App1.MainActivity->App1.SecondActivity-> App2.SecondActivity

App1.SecondActivity进程名：com.demo.app1:second

App2.SecondActivity进程名：com.demo.app2:remote2

②App1.MainActivity-> App1.ThirdActivity-> App2.SecondActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2.SecondActivity进程名：com.demo.app2:remote2

（2）multiprocess=true时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即被启动的activity能够在另一个app中被实例化？

①App1.MainActivity->App1.SecondActivity-> App2.ThirdActivity

App1.SecondActivity进程名：com.demo.app1:second

App2. ThirdActivity进程名：com.demo.app2:remote3

②App1.MainActivity-> App1.ThirdActivity-> App2. ThirdActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2. ThirdActivity进程名：com.demo.app2:remote3

#### 4.5.2.3 本节小结

签名不同的不同App的component：无论它们的process为何值，无论multiprocess为何值，组件被启动时都与启动组件的进程无关，总是运行在默认app进程中。

**4.5.3 启动和被启动的activity不在同一App中（签名相同）**

根据前边的实验结果，我们需要修改app2的签名与app1的签名相同，设置app2.shareduserid与app1.shareduserid相同。

#### 4.5.3.1 activity.process未设置

（1）multiprocess=false时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即是否总在默认的app进程中？

①App1.MainActivity->App1.SecondActivity-> App2.SecondActivity

App1.SecondActivity进程名：com.demo.app1:second

App2.SecondActivity进程名：com.demo.app2

② App1.MainActivity-> App1.ThirdActivity-> App2.SecondActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2.SecondActivity进程名：com.demo.app2

（2）multiprocess=true时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即被启动的activity能够在另一个app中被实例化？

①App1.MainActivity->App1.SecondActivity-> App2.ThirdActivity

App1.SecondActivity进程名：com.demo.app1:second

App2. ThirdActivity进程名：com.demo.app1:second

②App1.MainActivity-> App1.ThirdActivity-> App2. ThirdActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2. ThirdActivity进程名：com.demo.app1:third

#### 4.5.3.2 activity.process有值

<activity android:name=".SecondActivity"

android:label="App2-SecondActivity"

android:multiprocess="false"

android:process=":remote2">

</activity>

<activity android:name=".ThirdActivity"

android:label="App2-ThirdActivity"

android:multiprocess="true"

android:process=":remote3">

</activity>

（1）multiprocess=false时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即是否总在“:remote”进程中？

①App1.MainActivity->App1.SecondActivity-> App2.SecondActivity

App1.SecondActivity进程名：com.demo.app1:second

App2.SecondActivity进程名：com.demo.app2:remote2

②App1.MainActivity-> App1.ThirdActivity-> App2.SecondActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2.SecondActivity进程名：com.demo.app2:remote2

（2）multiprocess=true时，从多条路径启动activity，观察它所在的进程是否与启动它的组件所在进程有关？即被启动的activity能够在另一个app中被实例化？

① App1.MainActivity->App1.SecondActivity-> App2.ThirdActivity

App1.SecondActivity进程名：com.demo.app1:second

App2. ThirdActivity进程名：com.demo.app1:second

② App1.MainActivity-> App1.ThirdActivity-> App2. ThirdActivity

App1.ThirdActivity进程名：com.demo.app1:third

App2. ThirdActivity进程名：com.demo.app1:third

#### 4.5.3.3 本节小结

签名相同的不同App的component：无论process设置与否，multiprocess为“false”，组件被启动时与启动组件的进程无关，总是运行在默认app进程中；Multiprocess为“true”，组件被启动时与启动组件的进程有关。

**5 应用程序和组件权限配置研究**

我们可以使用<uses-permission>声明App具有哪些权限，也可以使用permission属性对App的使用者做限制。

**5.1 共享用户间APP的权限集合**

设APP1和APP2共享用户ID，即sharedUserId和签名都相同。APP1申请的权限集合是P1={P11,P12，P13}，APP2申请的权限集合P为P2={P11，P21，P22}。

（1）首先把APP1安装，观察APP1的权限集合是否为P1;在app1的Androidman- ifest.xml添加：

<!--打电话(危险权限)-->

<uses-permission android:name="android.permission.CALL\_PHONE"/>

<!--发短信(危险权限)-->

<uses-permission android:name="android.permission.SEND\_SMS"/>

<!-- 普通权限 -->

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />

安装app1，查看packages.xml中app1的package信息如图5-1所示：



**图5-1 packages.xml中App1的package信息**

可以看到只记录普通权限，不记录危险权限。危险权限在运行时动态获取[17]。

（2）在APP1已安装的条件下，安装APP2。若能成功安装，观察APP1和APP2的权限集合是否相等，而且都为P1∪ P2。在app2的Androidmanifest.xml添加：

<!-- 打电话(危险权限) -->

<uses-permission android:name="android.permission.CALL\_PHONE" />

<!-- 访问SD卡(危险权限) -->

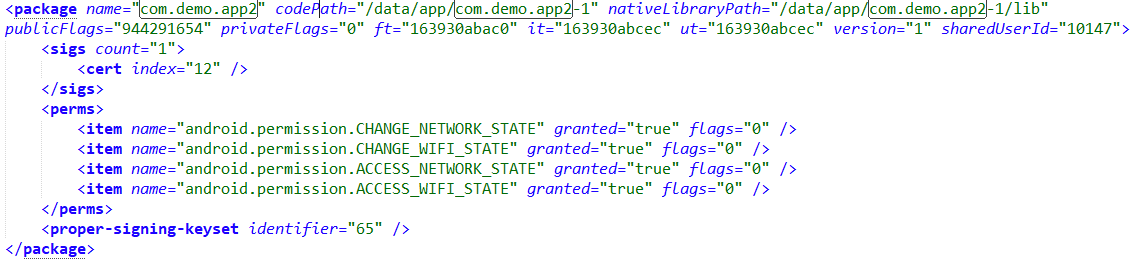
<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<!-- 普通权限 -->

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_NETWORK\_STATE" />

安装app2，查看packages.xml中app2的package信息如图5-2所示：



**图5-2 packages.xml中App2的package信息**

查看此时app1的package信息，如图5-3所示：



**图5-3 packages.xml中App1改变后的package信息**

可以看到app1的package信息发生变化，两个app的权限取了普通权限的并集。

危险权限在这里并不记录，需要我们在Android设备的权限管理中手动授予。

**5.2 enforced权限的继承**

由于application和activity中都可以通过permission属性声明enforced permissions，因此它们之间的关系就需要明确定义。该实验就是测试它们之间的关系。

制作两个APP：APP1和APP2。APP1中包含A1组件，APP2中包含A21，A22组件。要用A1启动A21和A22，即A1向A21和A22发送Intent。

实验前提：在做下列实验中，app1和app2的签名不同，shareduserID也不相同。

测试条件：

（1）APP1.<uses-permission>为空，A1.permission=null，APP2.permission=null,- A21.permission=null，A22.permission=null，A1启动A21，A1启动A22，观察A1发送操作能否成功？A21，A22能否启动成功？

①运行app1，点击button“App2:FirstActivity”：（启动成功）

②运行app1，点击button“App2:SecondActivity”：（启动失败，app1崩溃）

③运行app2，点击button“SecondActivity”和“ThirdActivity”：（启动成功）

（2）APP1.<uses-permission>为空，A1.permission=null，APP2.permission=P1, A21.permission=null，A22.permission=null，A1启动A21，A1启动A22，观察A1发送操作能否成功？A21，A22能否启动成功？

在项目app2的Androidmanifest.xml的application中添加：

<application

android:permission="android.permission.ACCESS\_NETWORK\_STATE">

</application>

①运行app1，点击button“App2:FirstActivity”：（启动失败，app1崩溃）

②运行app1，点击button“App2:SecondActivity”：（启动失败，app1崩溃）

③运行app2：（启动失败App isn’t installed.）

（3）APP1.<uses-permission>为空，A1.permission=null，APP2.permission=null, A21.perm- ission=P1，A22.permission=null，A1启动A21，A1启动A22，观察A1发送操作能否成功？A21，A22能否启动成功？

在项目app2的Androidmanifest.xml的SecondActivity中添加：

<activity android:name=".SecondActivity"

android:permission="android.permission.ACCESS\_NETWORK\_STATE"

></activity>

①运行app1，点击button“App2:FirstActivity”：（启动成功）

②运行app1，点击button“App2:SecondActivity”：（启动失败，app1崩溃，app1没有<uses-permission>和app2没有exported默认为false导致）

③运行app2，点击button“SecondActivity”和“ThirdActivity”：（启动成功）

（4）APP1.<uses-permission>=P1，A1.permission=null，APP2.permission=null, A21.permi- ssion=P1，A22.permission=null，A1启动A21，A1启动A22，观察A1发送操作能否成功？A21，A22能否启动成功？

在项目app1中Androidmanifest.xml中manifest标签下添加：

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

①运行app1，点击button“App2:FirstActivity”：（启动成功）

②运行app1，点击button“App2:SecondActivity”：（启动失败，app1崩溃，app2没有exported默认为false导致）

③在项目app2的Androidmanifest.xml中secondActivity中添加：

<activity android:name=".SecondActivity"

android:exported="true"

></activity>

再次运行app1，点击button：“App2:SecondActivity”：（启动成功）

④删除在③中添加的“android:exported="true"”，运行app2，点击button“Second- Activity”和“ThirdActivity”：（启动成功）

（5）APP1.<uses-permission>=P1，A1.permission=null，APP2.permission=null, A21.per- mission={P1，P2}，A22.permission=null，A1启动A21，A1启动A22，观察A1发送操作能否成功？A21，A22能否启动成功？

在项目app2的Androidmanifest.xml的SecondActivity中添加：（出错）

<activity android:name=".SecondActivity"

android:permission="android.permission.ACCESS\_NETWORK\_STATE"

android:permission="android.permission.CHANGE\_NETWORK\_STATE"

></activity>

直接出错，结果：组件下permission属性唯一。

（6）APP1.<uses-permission>=P1，A1.permission=null，APP2.permission=P1, A21.per- mission=P2，A22.permission=null，A1启动A21，A1启动A22，观察A1发送操作能否成功？A21，A22能否启动成功？

在项目app1中Androidmanifest.xml中manifest标签下添加：

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

在项目app2的Androidmanifest.xml的application中添加：

<application

android:permission="android.permission.ACCESS\_NETWORK\_STATE">

</application>

在项目app2的Androidmanifest.xml中secondActivity中添加：

<activity android:name=".SecondActivity"

android:permission="android.permission.CHANGE\_NETWORK\_STATE"

></activity>

①运行app1，点击button：“App2:FirstActivity”：（启动成功）

②运行app1，点击button：“App2:SecondActivity”：（启动失败，app1崩溃）

③运行app2，点击button：“SecondActivity”和“ThirdActivity”：（启动成功，但是app2只可以运行1次）。

**5.3 本章小结**

相同用户的App的普通权限取它们的普通权限合集，并记录在packages.xml文件中；危险权限不记录在packages.xml文件中，它在运行时动态获取或者手动授予，并且是按权限组进行获取的。如果应用请求系统危险权限，而应用在同一权限组中已有另一项危险权限，则系统会立即授予该权限，而无需与用户进行任何交互。

permission属性是对其他用户的使用做限制，当在同一用户下，不进行permission检查，即Shareuserid相同，同一用户情况下，两个app可以相互调用各组件，即使目标app2的component添加了permission，app1没有使用<uses-permission>进行声明，依旧可以调用，所以permission是对其它用户做限制。

**6 组件间通信的权限分析**

Intent(意图)在Android中是一个十分重要的组件，它是连接不同应用的桥梁和纽带。Intent的使用分为二个方面一个是发出Intent，另一个则是接收Intent。

**6.1 Intent的隐式启动**

Intent的显示启动就是直接指定需要启动的activity，可以唯一确定一个Activity，意图明确。隐式启动不明确指定启动哪个Activity，而是设置Action、Data、Category，让系统来筛选出合适的Activity，筛选是根据所有的<intent-filter>来筛选。

APP2.activity中有一个intentFilter，观察能否从APP1通过发送该intent启动APP2.activity。在项目APP2的Androidmanifest.xml中thirdActivity添加：

<activity android:name=".ThirdActivity">

<intent-filter>

<action android:name="com.demo.app2.thirdactivity\_MyAction" />

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

</intent-filter>

</activity>

运行APP1，点击button“App2:ThirdActivity”：启动成功。

**6.2由Intent action所施加的权限**

Intent可以看作一个二元组<action, data>，其中action是Intent期望执行的操作，data是数据。一个组件发送一个Intent时，可能需要发送权限，同样，一个组件用IntentFilter接收一个Intent时也可能需要相应的权限。

比如，当一个组件A拟发送android.intent.action.CALL时，它必须具有权限： android.permission.CALL\_PHONE。当一个组件拟接收android.intent.action.PHONE\_ST- ATE时，它必须具有权限： android.permission.READ\_PHONE\_STATE。

本小节就是测试Intent发送和接收对权限均有相应要求这一事实。A是APP的一个Activity。测试条件，application.<uses-permission>为空：A发送一个action为android.intent.- action.CALL的Intent，观察是否能发送成功（否，程序奔溃）。

ThirdActivity.java中onCreate()函数修改FourthActivity的监听器：

Button button\_fourth\_activity = (Button)findViewById(R.id.FourthActivity);

button\_fourth\_activity.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

String tel\_num = "10086";

Uri tel\_call = Uri.parse("tel:"+tel\_num);

Intent intent = new Intent("android.intent.action.CALL", tel\_call);

startActivity(intent);

}

});

运行FourthActivity：（出现错误，提示“Permission Denial”）。

application.<uses-permission>= android.permission.CALL\_PHONE，A发送一个action为android.intent.action.CALL的Intent，观察是否能发送成功（可以），添加user-permission只是声明权限，因为它是危险权限，需要动态获取，如果此时不授予APP此权限，运行后还是出错，需要在设置（权限管理）中手动授予其权限。

运行FourthActivity：（启动成功）。

**6.3 本章小节**

在前边我们知道要跨应用启动activity，需要设置exported属性为“true”，如果定义了至少一个intentFilter，那么exported的默认值为true，如果没有定义intentFilter，则默认值为false。在实际开发中，我们应当注意如果不想让其他应用启动该activity，需要再次将exported设置为“false”，以此减小安全隐患。

除此之外，需要将permission要求的权限与action间接需要的权限区分开来，permission属性值是要求发送者所要具有的权限，action则是应用本身访问此action需要具有的权限，即自身所要具有的权限。

**7 总结**

本文介绍了Android平台的发展状况以及在发展过程中面临的安全威胁，并对当前国际的研究现状进行了介绍，由于恶意APK软件的不断增多，为了保证Android应用程序的安全性，必须充分利用Android系统提供的权限机制。尽管Android官方文档已经对配置文件中的各种配置项进行了说明，但是有些说明没有给出精确地语义描述，有些说明与实际开发甚至是矛盾的，有些说明没有给出相关配置项之间的相容、互斥关系，由此引出本课题的研究意义。在进行本课题的时主要对如何配置应用程序（包括组件）与进程之间的关系、如何配置应用程序（包括组件）的权限、如何配置组件间通信的权限约束关系。采用的方法是，在阅读官方文档的基础上，对存在语义模糊的说明，设计测试用例，开发测试程序，通过程序的运行结果，印证并进一步明确相关配置项的准确含义，并在此基础上总结出若干配置规则，以供设计和开发人员使用，提高开发效率，减少程序安全漏洞。

**致 谢**

美好的大学生涯即将以这最后几个月的毕业设计而宣告结束，这是大学最后的作业。这四年的大学，因为学校，因为老师，我可以容易的学到教程要求的知识，因为学校给予我一个广阔自由的平台，老师的授课欢快又不失严肃，这种环境使我可以在学校安心的充实自己，丰富自己。同学校友间的融洽相处，互帮互助让我心里倍感到温暖。一个离家求学的人仿佛又找到了自己的新家如此的温暖。学会了如何去关心他人，帮助他人，正确的为人处事的方式积极乐观的生活态度。我还学会了学习一切重在自己的学习方法，这会让我在以后的工作中受益匪浅。

在这里感谢我的指导老师刘晓建老师一步一步耐心的指导和讲解，感谢我的同学宋磊在完成本次课题过程中给予我的帮助。同时，我也非常感谢我的室友和同学们，他们给了我很大的关心和支持。陪伴我走过了四年的大学生活。

最后，希望自己能够以这份毕设完美的结束自己的学生生涯，再用激情去迎接更加充满希望而美好的明天。

**参考文献**

[1] William Enck, Machigar Ongtang, Patrick McDaniel. Understanding Android Security[J],2009

[2] 余丽芳. 基于Android系统组件安全通信的研究[D].北京:余丽芳,2017.

[3] 郭霖. Android第一行代码[M].北京:中国工信出版集团,2016

[4] 闫梅. Android应用程序权限检测机制的研究[C].山西:闫梅,2015

[5] 李晓洲. Android应用程序组件漏洞测试方法研究[C].山西:李晓洲,2012

[6] William Enck, Damien Octeau, Patrick Mc Daniel, etc. A Study of Android Application Security[C]. Proceedings of the 20th USENIX Conference on Security. USA:USENIX Association, 2011

[7] P.J.Denning. Fault to lerant operating systems. ACM Computing Surves(CSUR). 1976

[8] 彭凌. Android应用权限检测技术的研究与改进[C].北京:彭凌,2015

[9] 廖明华,郑力明. Android 安全机制分析与解决方案初探[J].科学技术与工程,2011

[10] 龚利. 浅析组件安全对Android系统的影响[J]. 郧阳师范高等专科学校学报,2015

[11] 丰生强.Android软件安全与逆向工程[M].北京:人民邮电出版社,2013,2,156-157

[12] 柯元旦.Android内核剖析[M].北京:电子工业出版社,2012,20-23

[13] 传智播客高教产品研发部.Android移动应用基础教程[M].北京:中国铁道出版社，2014,4-5

[14] 杨丰盛.Android技术内幕系统卷[M].北京:机械工业出社,2011,56-62

[15] 狄婧.Android安全机制分析与解决方案解析[J].硅谷,2011,24:60-62

[16] 张中文,雷灵光,王跃武.Android Permission机制的实现与安全分析[C].第27次全国计算机安全学术交流会.2012.8:3-5

[17] A.Shabtai.Malware Detection on Mobile Devices[C].Proceeding of the 11th International Conference Monbile Data Management.Kansas City,Mission,USA,

2010:289-290