Android 虚拟设备权限的设计

1、前言

Android系统是目前最流行的移动平台操作系统之一，它继承了Linux内核的安全机制，同时也具备一套自身特色的安全机制。在Android 6.0版本之前，当用户在安装应用程序时，AndroidManifest清单上申请的权限都会被系统默认授权，并且在安装授权后无法撤销。这样存在着一定的弊端，因为有些权限用户是在真正使用到才会觉得不需要，比如通知权限，用户在运行应用程序时不想受到通知的干扰，就是不需要通知权限，但用户安装时已经授权，此时却无法撤销。还有一些情况是，一些恶意程序，会利用这个权限默认授权，进行恶意获取用户数据和攻击，但用户无法撤销。因此谷歌在2015年8月份时候，发布的Android 6.0版本，代号“棉花糖”（Marshmallow ），其中的很大的一部分变化，是在用户权限授权上。在6.0版本中，引入了运行时权限，就是只有在用户需要使用权限的时候，才去授权请求，而不是安装时全部授予。Android6.0的变化，在一定程度上提高了安全性，但是在另一方面却增加了权限的碎片化，增加了开发者的工作量，并且衍生出另外一个问题，应用程序如果不具备某一特定权限，则会导致程序强制退出，无法运行，这在一定程度上影响了用户的体验度。本文基于Android系统已有的权限管理机制，结合虚拟设备的添加，研究Android虚拟设备添加的权限设计。

2、系统设计

2.1 Android权限管理机制的整体架构

Android系统在MarshMallow之前，权限都是在安装的时候授予的，虽然在4.3时，Google就试图在源码里面引入AppOpsManager来达到动态控制权限的目的，但由于不太成熟，在Release版本中都是把这个功能隐藏的。在6.0之后，Google为了简化安装流程且方便用户控制权限，正式引入了runtime-permission，允许用户在运行的时候动态控制权限。

Android6.0之后，权限主要分为normal、dangerous、SYSTEM\_ALERT\_WINDOW 和 WRITE\_SETTINGS四个等级，其中常用到的是正常权限和危险权限，如下图。

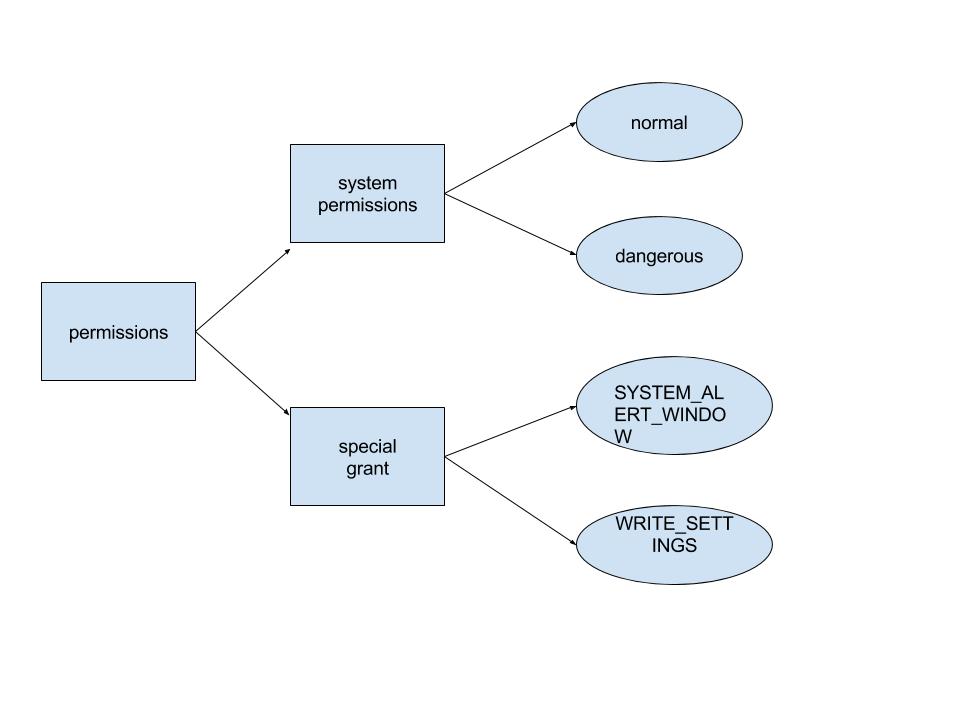


图1 权限类型

normal：正常权限涵盖应用需要访问其沙盒外部数据或资源，但对用户隐私或其他应用操作风险很小的区域。应用声明其需要正常权限，系统会自动授予该权限。例如设置时区，只要应用声明过权限，系统就直接授予应用此权限。dangerous：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。例如读取用户联系人，在6.0以上系统中，需要在运行时明确向用户申请权限。

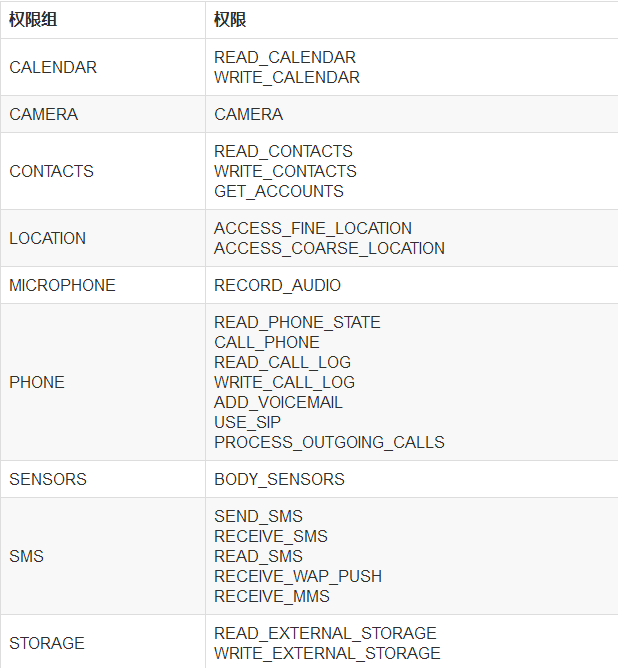


图2、dangerous权限分组

2.2 Android权限管理的工作流程

Android6.0的runtime-permission机制让用户在任何时候都可以取消授权，只有在需要权限的时候，才告知用户是否授权，是在runtime时候授权，而不是在原来安装的时候 ，同时默认情况下每次在运行时打开页面时候，需要先检查是否有所需要的权限申请。这样的用户的自主性提高很多，比如用户可以给APP赋予摄像的权限，但是可以拒绝记录设备位置的权限，防止位置信息上传等等。

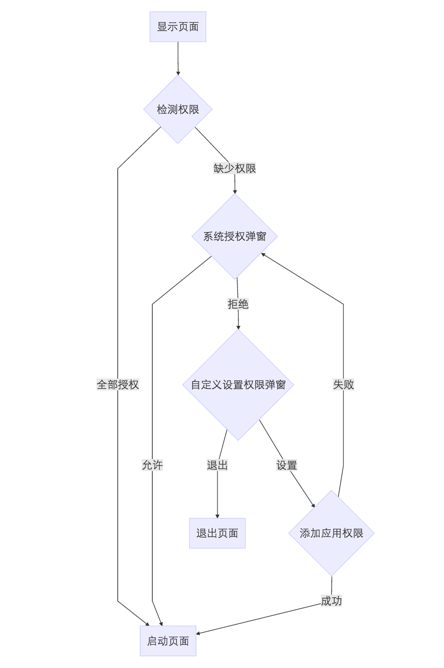


图3 权限流程图

因此，每次在申请系统服务的时候，都要动态查询是否获取了相应的权限，如果没有获取，就需要动态去申请，这时会有弹出一个系统标准Dialog提示申请权限，如下图：

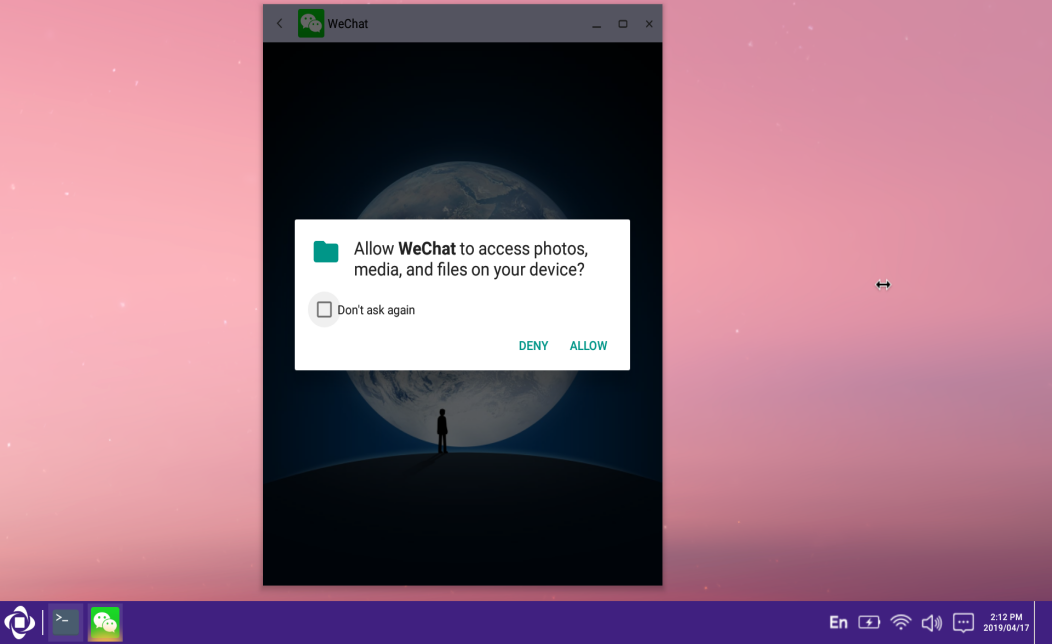


图4 权限授予提示框

用户选择授权或者拒绝后，AppOps会将选择记录在案，并通知申请服务提供或者拒绝服务。

2.3 AppOps子系统

AppOps全称是Application Operations，是Android原生的应用程序权限管理，类似于平常所说的应用程序的操作（权限）管理。AppOps是Google原生Android包含的功能，但是Google在每次版本更新时都会隐藏掉AppOps的入口。

AppOps的核心服务是APPOpsService，实现了大部分的核心功能逻辑，但它不能被其他模块直接调用访问，而是通过AppOpsManager提供访问接口。AppOpsManager实现的动态管理的本质是：将鉴权放在每个服务内部，比如，如果App要申请定位权限，定位服务LocationManagerService会向AppOpsService查询是否授予了App定位权限，如果需要授权，就弹出一个系统对话框让用户操作，并根据用户的操作将结果持久化在文件中，如果在Setting里设置了响应的权限，也会去更新相应的权限操作持久化文件/data/system/appops.xml，下次再次申请服务的时候，服务会再次鉴定权限。

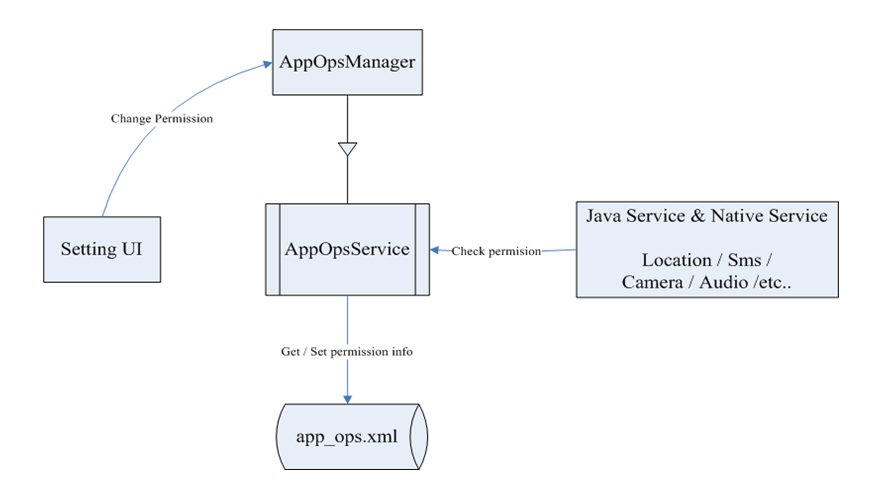


图5 AppOps整体的工作框架图

2.4 PKMS子系统

Android的runtime-permission机制让用户在任何时候都可以取消授权，因此，每次在申请系统服务的时候，都要动态查询是否获取了相应的权限，如果没有获取，就需要动态去申请。上一节当中说到AppOps会进行权限的检查，其实际是使用PKMS来完成最终的权限验证。PackageManagerService是一个负责处理各种应用程序的安装、卸载、管理等工作的服务，而权限管理的APP和安装器是同一个PackageInstaller。PackageInstaller负责应用的安装与卸载，里面还包含了对授权管理的一些逻辑，PackageInstaller中的GrantPermissionsActivity主要负责权限的授予工作。在这个Activity里面，如果一开始没有获得权限，就会弹出权限申请对话框，根据用户的操作去更新PKMS中的权限信息，同时将更新的结构持久化到runtime-permissions.xml中去。这些持久化的数据会在手机启动的时候由PMS读取,开机启动，PKMS扫描Apk，并更新package信息，检查/data/system/packages.xml是否存在，这个文件是在解析apk时由writeLP()创建的，里面记录了系统的permissions，以及每个apk的name,codePath,flags,ts,version,uesrid等信息，这些信息主要通过apk的AndroidManifest.xml解析获取，解析完apk后将更新信息写入这个文件并保存到flash，下次开机直接从里面读取相关信息添加到内存相关列表中，当有apk升级，安装或删除时会更新这个文件，packages.xml放的只包括installpermission，runtimepermissiono由runtime-permissions.xml存放。

2.5 VIVID

Android系统中经常涉及到具体硬件的访问，比如说camera、GPS、Audio、Storage等等，这些硬件的权限基本都属于dangerous权限，需要在运行时向系统请求授予，否则应用程序无法继续运行。以camera为例，在Android新的安全机制runtime-permission中，如果应用程序需要使用camera权限却没有被授予，则会导致程序crash。为了保护用户隐私同时兼顾应用程序的正常运行，引入了虚拟设备的添加。在AppOps及PKMS中会涉及到权限的检查，以Camera为例，如果应用申请了Camera权限且被授予运行时权限，此时可以正常访问物理摄像头；如果应用申请了Camera权限但没有被授予运行时权限，此时应用程序不会退出，但是获得的数据来自虚拟摄像头。这在一定程度上解决了权限管理与物理设备、虚拟设备的融合，提高了系统安全性，也增加了用户体验度。