计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(7):238−244 [doi: 10.15888/j.cnki.csa.005836]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Android开发中利用反射获取存储路径的研究①

陆志平, 胡晨骏

(南京中医药大学信息技术学院, 南京 210046)

摘　要: Android应用的开发中对存储的访问非常频繁, 但是Android各个版本对存储的支持比较混乱, 部分版本甚至没有公开的API支持对扩展存储访问. 对Android的内置、外置存储设备进行研究后, 提出将存储分为内部存储、外部存储及扩展存储三个类型. 分析了各个存储的类型的特点及访问方式, 重点讨论了扩展存储的访问方式, 提出利用JAVA的反射机制来获取Android平台的扩展存储目录, 解决了Android不同版本对存储进行访问的兼容性问题. 通过分析工具分析了反射机制在此应用中的效率问题, 并在不同Android版本的设备上进行了测试.

关键词: Android; 反射; 内部存储; 外部存储; 扩展存储; 路径

引用格式: 陆志平,胡晨骏.Android开发中利用反射获取存储路径的研究.计算机系统应用,2017,26(7):238–244. http://www.c-s-a.org.cn/1003-3254/5836.html

Research on Android Developing and Utilizing Reflection to Gain Storage Path

LU Zhi-Ping, HU Chen-Jun

(Information and Technology College, Nanjing University of Chinese Medicine, Nanjing 210046, China)

Abstract: In the development of Android application, the access to the storage is extremely frequent, but different versions of Android are in a muddle in supporting storage, with some versions of Android even without public API support for extended storage access. After conducting a study on Android's built-in and external storage devices, we propose to divide storages into three types: internal storage, external storage and extended storage. What's more, we have analyzed the characteristics and access modes of various types of storage, especially the extended storage access mode. We use JAVA reflection mechanism to obtain extended store directory of Android platform. We solve the compatibility problem that different versions of the android access the storage device. Through the analysis tool we have analyzed the efficiency of reflection mechanism in this application, and has tested it on multi-model equipment.

Key words: Android; reflection; internal storage; external storage; extended storage; path

Android是一种基于Linux的开放源代码的操作系统, 被大量移动设备所采用. 自2008年发布第一个版本以来, 历经多年, 目前已经发展到6.0版本. 在早期的2.x版本中, Android设备都是单存储设置, 访问存储设备的路径比较统一, 也有API支持存储设备的访问. 而到了4.0版本之后, Android将存储分为了primary storage和secondary external storage, 一些设备生产商又会扩展多个存储设备. 例如现在很多手机不仅支持外置SD卡, 还支持外置U盘, 导致存储管理相当混乱. 而Android也提倡少用外置的扩展存储, 所以将访问外置存储的API隐藏了起来, 不作为公开的API. 目前在项目的开发中, 访问外置存储会经常用到. 项目开发必须充分考虑程序的健壮性, 且要能兼容多个版本的Android系统, 所以有必要对存储路径的获取做一个深入的研究, 找出一个合适的方法.

获取扩展存储路径的有多种方法, 例如如下几种方法; 过滤Android系统中"mount"命令的结果, 但是"mount"命令的结果会把一些具有特殊作用目录的挂

① 基金项目: 国家自然科学基金(61003180); 江苏省产学研联合创新资金(BY2013063-08); 南京中医药大学青年自然科学基金(13XZR34)
收稿时间: 2016-10-22; 收到修改稿时间: 2016-12-01

载信息也包含进去, 这样就很难区分存储设备, 容易判断错误. 也可以利用Android系统中“vold.fstab”文件内的信息来判断存储信息, 但是“vold.fstab”文件在不同的设备上, 存储的目录也不尽相同, 查找起来相当麻烦. 在大部分设备上也可以通过过滤Android的环境变量“SECONDARY_STORAGE”来找到扩展存储的目录[1], 但环境变量“SECONDARY_STORAGE”在某些设备上没有, 所以上述方法虽然在大部分设备上能够访问到扩展存储目录, 但也存在一些缺陷.

基于Android并没有将访问扩展存储的API废弃, 而是将其隐藏起来, 所以本文研究采用Java的反射机制来读取Android中隐藏的API, 通过隐藏的API来访问扩展存储的目录, 并采用市场上不同Android版本的常用设备进行测试验证.

1 Java的反射机制

1982年Brian Cantwell Smith在他的博士论文中提出了反射的概念, 提出反射是计算机程序在运行时可以访问、检测和修改它本身状态或行为的一种能力[2]. 这一概念的提出引发了反射在程序设计方面应用的研究, 很快被很多程序设计语言所采用, 例如Java、C#、Ruby、PHP、Perl等语言都支持反射机制.

在Java中, 程序在运行状态下, 通过反射机制可以动态地获取任意一个类的所有属性和方法, 能调用任意一个对象的方法与属性[3].

Java程序若要正常运行, 需要Java虚拟机加载相应的Java类[4]. 一般情况下, 程序运行时所使用的相关类在编译期就已经加载了. 但是在程序运行期间, 若是需要一个新的类来完成某些操作时, 就需要重新编译. 而采用反射后, Java就可以在编译期间不知道类或接口的名称、成员变量、方法的情况下, 在运行时检查类、接口的成员变量和方法, 并允许实例化此类的对象并调用其方法[5].

Java中使用反射机制通常所需要的的类有: Class、Field、Constructor、Method等[6], 其中, Class类是Java反射机制的基础.

在Java中, 某个将要操作的类被加载后, Java虚拟机(JVM)就会自动生成一个Class类型的对象, 此Class对象描述了加载到虚拟机中的类对应的成员变量、构造方法以及其他方法等相关信息[7]. Field、Constructor、Method类则分别对应为加载类的成员变量、构造函数以及方法的抽象.

由于Class类的构造方法的访问属性是private, 因此不能直接通过构造方法来新建一个Class对象, 若需要获取操作类的Class对象, 一般通过如下3种常用方式[8].

1) 使用对象的getClass方法.

2) 使用类的.class语法.

3) 使用Class类的静态方法forName().

通过上述方法获得Class对象之后即可以访问操作类自身的信息, 例如此操作类隶属的Package、类别的访问属性、继承类、实现接口、访问此操作类的成员变量及方法等.

利用Java中提供的reflection API在程序运行的时候获取类信息的方法如下.

1) 获取类的构造方法

Constructor<T> getConstructor(Class<?>... parameterTypes)

Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)

上述两个方法通过指定参数列表获得特定的Constructor对象, 前者返回访问属性为公共的构造方法, 后者返回公共、保护、默认和私有这几种访问类型的构造方法.

Constructor<?>[] getConstructors()

Constructor<?>[] getDeclaredConstructors()

上述两个方法返回一个包含Constructor对象的数组, 同理, 前者返回数组中的构造函数对象的访问属性只能为public.

2) 获取类的其他方法

Method getMethod(String name, Class<?>... parameterTypes)

Method getDeclaredMethod(String name, Class<?>... parameterTypes)

Method[] getMethods()

Method[] getDeclaredMethods()

与获得类的构造方法类似, 上述四个方法可以获得类的其他方法.

3) 获取类的成员变量

Field getField(String name)

Field getDeclaredField(String name)

Field[] getFields()

Field[] getDeclaredFields()

同理, 上述四个方法用于获得类的成员变量, 前两

个返回单个成员变量, 后两个返回成员变量对象构成的数组.

从上面三类方法可以看到, 返回的类的各种信息分别以Constructor、Method以及Field对象的形式存放. 在程序中, 可以通过调用这些对象中相应的方法来动态的创建对象、调用方法以及获取成员变量.

2 内部存储、外部存储及扩展存储

如今Android设备都会内置一块大容量的存储卡, 例如eMMC卡(Embedded Multi Media Card). 一些手机的16 G/32 G版本, 指的就是此手机内置卡的容量. 除了此内置存储卡, 很多Android设备还额外配置了扩展存储设备, 例如Micro SD卡(Micro Secure Digital Memory Card)、USB设备等[9]. 如图1所示.

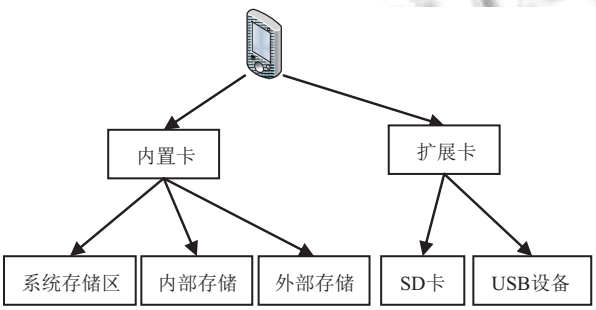


图1 Android设备存储结构图

2.1 内部存储

Android设备的内部存储并不是传统意义上的内存, 内存指的是设备运行时所用数据的存储区, 与计算机中的内存一样. 而Android的内部存储是用来安装各类应用程序, 或者用来保存应用程序的相关数据, 是一个数据的长久存储区.

内置存储卡的一部分存储空间被划分出来存放系统文件, 此分区由于安全等方面的考虑, 一般是隐藏的, 用户通过普通的文件管理软件是不能访问的, 此分区存放着"/system"、"/cache"等系统目录. 一部分存储空间被划分出来用来安装各类应用程序与保存相关数据, 此存储区称为内部存储, 也是隐藏的, 目录为"/data"[10].

在内部存储中, 应用程序创建后, 在/data/data目录下会创建一个与应用程序包名相同的一个目录, 此目录用来存放同名应用程序的相关文件, 且默认只能被此应用程序访问. 此机制保证了各个应用程序之间数据的独立性与安全. 若是应用程序被卸载, 则内部存储中与应用程序包名相同的目录及目录内的数据也会被删除.

2.2 外部存储

内置存储卡中剩下存储空间的用来保存各类用户数据, 称为为外部存储. 外部存储区是公共的, 可以通过普通的文件管理软件进行访问与管理[11].

相对于内部存储, 外部存储可以用来保存用户的各种数据, 且本存储区的数据为可访问数据. Android设备中的某些文件需要被各类应用程序调用, 所以此类文件通常为共享类型. 内部存储一般不存储此类文件, 此时可以将此类文件存放在外部存储中, 例如照相机拍摄的相片、录制的视频和音频等文件. 用户也可以在外部存储中建立新的文件、目录, 也可以对外部存储中的文件或文件夹做各种修改.

应用程序也可以在外部存储存储数据. 安装了应用程序后, 在内部存储"/data/data"目录下相应的包名目录下存放了此应用的相关数据; 而在外部存储中, 也有一个"/Android/data/ Package_Name"的目录, 此目录可以存储与应用相关的数据, 如在线视频点播、在线音乐等软件的缓冲数据或者下载的文件就可以存放在此目录下. 例如在华为手机的大部分型号上, 应用安装后, 外部存储的"storage/emulated/0/Android/data/Package_Name" 目录中存放了应用的相关信息, 且应用程序删除后, 此目录的所有文件也会被自动删除, 这一特性与内部存储类似. 所以, 从文件管理角度来看, 若是应用程序需要在外部存储进行数据存储, 建议将数据都放在此目录中.

2.3 扩展存储

内部存储一般用于安装应用软件, 而外部存储空间有限, 面对如今越来越大的应用程序与大量的数据, 扩展Android设备的存储空间是一个必须面对的问题. 扩展的SD卡与USB设备是一个较为理想的方案.

早期的Android2.x版本中由于没有内部存储与外部存储的区别, 用户的数据都需要存储在扩展卡中, 所以访问方式比较统一, 但是文件的组织管理却比较混乱. 而苹果公司的移动电话与平板电脑是不支持外置扩展卡的. 出于安全性与文件管理等方面考虑, 谷歌在某些Android版本中也考虑采用此种模式, 在Android4.x版本中, 系统没有支持直接访问扩展SD卡, 例如此版本的谷歌Nexus手机就不支持SD卡扩展[12], 并且在Android系统中将访问扩展存储的方式设置成为隐藏函数. 目前有一些Android平台的设备也在尝试此类模式, 仅支持内置的存储卡, 但绝大部分Android设备提供了外置

存储卡的扩展口. 所以, 如何读取Android设备内置存储卡与扩展存储卡的目录将是本文要研究的一个重要内容.

3 存储目录及访问

3.1 内部存储的目录及访问

内部存储中存放着应用程序的相关数据, 其目录为“/data”, 可以通过Context类和Enviroment类提供的方法来进行访问. 例如下面列举了部分常用方法.

Context类提供的方法:

◇ File getDir(String name, int mode):返回“/data/data/Package_Name/”目录下的名称为name的File对象, 如果该文件夹不存在, 则创建一个新的名为name的文件夹.

◇ File getCacheDir():返回路径为“/data/data/Package_Name/cache”的File对象.

◇ File getFilesDir():返回路径为“/data/data/Package_Name/files”的File对象.

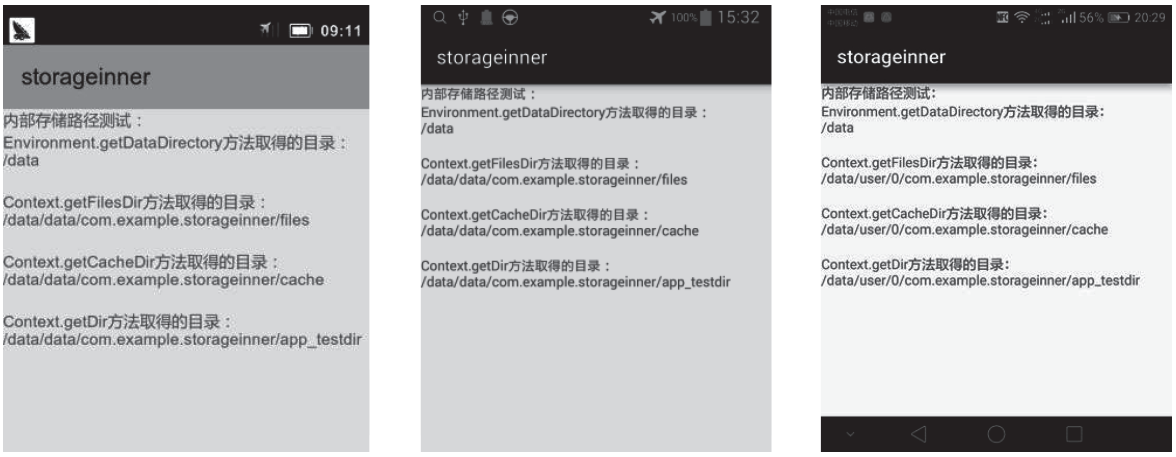
◇ String[] fileList(): 返回“/data/data/ Package_Name/files”目录下所有文件的文件名.

Environment类提供的方法:

◇ File getDataDirectory(): 返回路径为“/data”的File对象.

◇ File getDownloadCacheDirectory(): 返回Android下载/缓存内容目录.

应用部署到三款不同的设备上, 具体型号如图2所示, 可以看到通过上述方法都可以顺利的访问到内部存储的主目录及其子目录. 由于应用程序访问的数据都大都在目录“/data/data/ Package_Name/files”下, 若每次访问都要通过上述函数得到目录文件, 然后再构建输入输出流, 显然是比较麻烦的, 而Context类中提供了两个方法可以便捷的得到此目录的输入输出流. Context.openFileInput(String fileName) 和Context.openFileOutput(String fileName, int mode)这两个方法分别用于得到“/data/data/Package_Name/files”目录下文件名为fileName的输入流(FileInputStream)与输出流(FileOutputStream)[13].



HTC， Android2.3.3　　三星note2，　　华为mate7，

图 2　内部存储目录访问测试

3.2 外部存储的目录及访问

与内部存储类似, Context类和Enviroment类也提供了相应的方法来访问外部存储. 例如下面列举的部分常用方法.

Context类提供的部分方法:

◇ File getExternalFilesDir(String type): 返回的File对象为外部存储中某个应用程序存放私有文件的绝对路径, 比如上面提到的华为手机, 通过本函数得到路径为“storage/emulated/0/Android/data/Package_Name/files”的File对象. 由于目录存放于外部存储中, 所以即使是应用程序的私有文件, 它也能被其他应用程序访问. 且目录中的文件在应用程序卸载后也会被自动删除.

◇ File getExternalCacheDir(): 与getExternalFilesDir类似, 返回的File对象为外部存储中某个应用程序缓存文件的绝对路径.

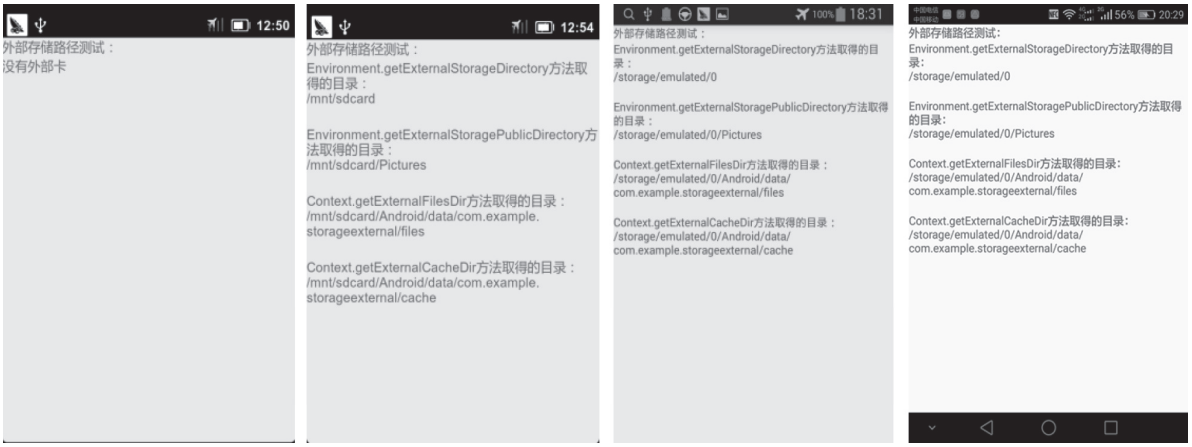
Environment类提供的部分方法:

◇ File getExternalStorageDirectory(): 返回外部存储根目录的File对象.

◇ File getExternalStoragePublicDirectory(String type): 返回外部存储中public文件夹下的File对象.

如图3所示为华为、三星等手机通过上述方法得到的外部存储的目录.

由于Android操作系统是一个开源的操作系统, 允许其他厂商对其进行深度定制, 所以导致android设备的外部存储的目录有多种版本; 例如一些图3中HTC的外部存储目录为“/mmt/sdcard”, 而Android4.0开始的外部存储目录为“/storage/emulated/0”, 另外还有其他一些品牌设备的外部存储目录也不一样, 这里就不一一列举. 所以外部存储的目录不可以使用固定的字符串, 而应该通过上述方法取得.



HTC，Android2.3无sd卡　HTC，Android2.3有sd卡　三星note2，Android4.3　华为mate7，Android6.0

图 3　外部存储目录访问测试

这里需要注意的是Android4.0版本之后, 才将存储分为了primary storage和secondary external storage, 所以对于4.0以前的版本来说, 外置的SD卡即外部存储. 所以上述方法即可读取到所有的存储路径, 不需要采用反射技术读取隐藏的API.

3.3　扩展存储的目录及访问

在Android平台上, 扩展存储的访问相对于内部存储与外部存储来说复杂的多. 考虑到Android开发中需要兼容各个版本, 所以, 设计一个兼容性好的访问方式是非常重要的.

通过JAVA反射机制来获取Android扩展存储目录的流程如下.

(1) 通过Android的API提供的getSystemService方法取得StorageManager类, 此类用于管理Android设备的扩展存储器.

(2) 通过反射机制获得StorageManager中的getVolumeList方法, 因为此方法为隐藏方法, 所以需通过Java的反射机制才能获取.

(3) 通过反射机制获得StorageVolume类, 因为StorageVolume类为隐藏类, 所以需通过反射才能得到StorageVolume的Class对象.

(4) 通过反射机制执行getVolumeList方法, 执行此方法后得到一个StorageVolume数组对象, StorageVolume数组对象中封装了设备的存储信息, 例如存储的挂载路径, 挂载状态及是否可以移除等.

(5) 通过反射得到StorageVolume类的getPath等方法, 并对StorageVolume数组对象执行. 例如执行getPath方法获得String类型的扩展存储目录、getPathFile方法返回File类型的扩展存储目录.

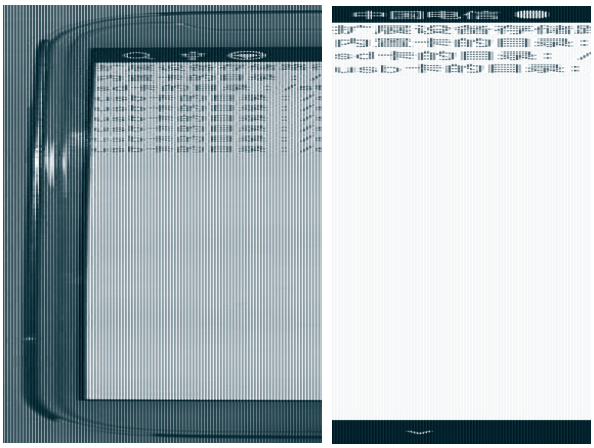
实施上述步骤的关键代码如下所示:

```
//通过Context类的getSystemService方法取得StorageManager对象
StorageManager mStorageManager = (StorageManager) mContext.getSystemService (Context.STORAGE_SERVICE);
Object[] storageVolumeobj[i]=null;
//通过反射机制得到getVolumeList方法
Method getVolumeList = mStorageManager.getClass().getMethod(“getVolumeList”);
//通过反射机制执行getVolumeList方法得到StorageVolume数组storageVolumeobj
storageVolumeobj=(Object[])getVolumeList.invoke(mStorageManager);
//通过反射得到StorageVolume的Class对象
```

```
storageVolumeClass = Class.forName("android.os.storage.StorageVolume");
//通过反射得到StorageVolume类的getPath方法, 执行本方法即可得到存储的目录
Method getPath = storageVolumeClass.get Method ("getPath");
//通过反射执行getPath方法获取外置卡路径
String path=(String) getPath.invoke(storage Volumeobj[i]);
//通过反射得到StorageVolume类的isRemovable方法
Method isRemovable=storageVolumeClass.getMethod("isRemovable");
//执行isRemovable方法, 判断到存储设备是否可移除.
Boolean is_removale = (Boolean) isRemovable.invoke(storageVolumeobj[i]);
```

上述代码中的isRemovable方法用来判断存储设备是否可移除, 内置卡不可移除, 而扩展存储着是可移除的, 从而判定扩展存储的目录.

从图4可以看到, 两款设备不仅插入了SD卡, 还通过连接线连接了USB设备. 通过上面介绍的反射机制, 均可以顺利的获取设备的内置卡的根目录, 即上文所说的外部存储, SD卡的目录以及USB设备的目录.



三星note2，Android4.3　　华为mate7，Android6.0

图4　反射获取外部存储和扩展存储的目录

4　兼容性分析与效率测试

鉴于目前Android设备型号复杂, 所采用的Android版本也各不相同, 并且由于Android平台开放的特性, 众多厂商会对其进行各种深度定制, 从而导致支持的存储设备也种类繁多. 所以如引言部分所述, 开发中对程序的健壮性与兼容性需要重点考虑.

本文提出的利用反射技术获取存储路径的方法, 并按此方法编写相应的程序在多台Android设备与Android版本上进行了详细的测试. 测试的部分结果如表1所示.

表1　读取存储路径测试结果表

Android版本	测试设备	内部存储	外部存储	扩展存储
2.3	HTC	API访问	API访问	无
4.3	三星Note2/华为荣耀6	API访问	API访问/反射机制访问	反射机制访问
6.0	华为Mate7	API访问	API访问/反射机访问	反射机制访问

表1仅列举了目前市场上较为常见的几种Android设备与Android版本的测试结果, 实际还在更多品牌的Android手机与平板电脑上进行了测试, 均能顺利运行, 并获取到了设备的各类存储目录.

从测试结果可以看到, 由于Android4.0版本之前为单存储设置, 所以采用系统提供的API即可解决存储的访问问题. 4.0版本之后, Android将访问扩展存储卡的API设置为隐藏属性, 但均没有取消掉, 可以通过反射机制获取隐藏的类、方法等信息, 所以采用反射机制获取存储路径的方法在各个Android版本及设备上均可获得支持, 在健壮性及兼容性方面均表现良好.

由于反射需要将内存中的对象进行解析, 包括了一些动态类型, 所以JVM无法对这些代码进行优化. 因此, 反射操作的效率要比非反射操作低. 而本文所提出的方法为利用反射技术来获取存储路径, 所以需要考虑到效率上对系统的影响, 下面通过TraceView[14]进行分析.

从图5可以看到各个方法所花费的时间, 其中, Name列为执行方法的名称, InclRealTime列为方法本身执行所花费的时间, InclRealTime%列为此方法运行时间所占的百分比. Call+RecurCalls/Total列为方法调用次数及递归调用占总调用次数的百分比. 从Name列可以看到, Envoronment.getxxx()是静态获取内部存储目录的方法, MainActivity.getStoragePath()为通过反射获取扩展卡目录的函数, 此方法耗时4.194毫秒, 在所有方法的耗时数中占了3.9%, 与部分静态获取路径的方法比较起来, 执行的效率并没有很大的差距, Class.forName()、

Class.getMethod()、Mehtod.invoke()为反射机制的应用, 从图中可以看到, 执行所占的时间比分别为0.2%、0.8%, 0.2%, 由于调用的次数不是很多, 所以对整个系统来说, 负载并不是特别大.

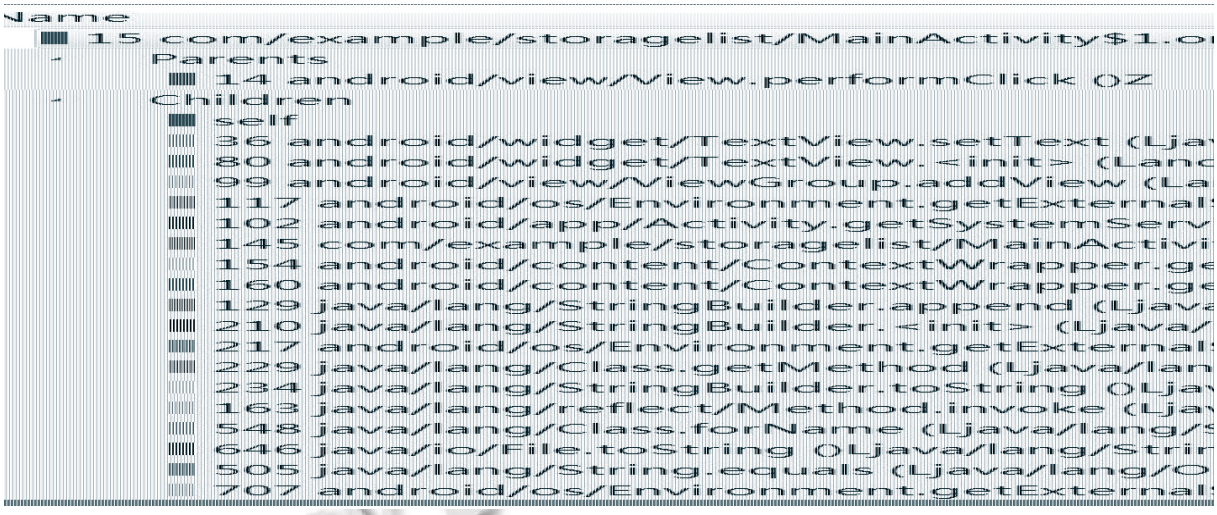


图 5 TraceView分析图

在设计应用时, 对存储目录的读取虽然频繁, 但基本不会用于次数庞大的循环程序中. 所以, 采用反射机制读取存储的目录对应用程序来说是完全可以的.

5 结语

综上所述, 采用本文所介绍的方法的应用在各个Android版本的设备上均可以正常高效运行, 顺利读取其存储目录, 且能正确获取内部存储路径、外部存储路径、扩展卡与USB设备的路径, 在健壮性、兼容性等方面均能满足需求.

参考文献

1 Kim H, Agrawal N, Ungureanu C. Revisiting storage for smartphones. ACM Trans. on Storage (TOS), 2012, 8(4): 14.

2 Smith BC. Reflection and semantics in a procedural language. Boston, MA: The Massachusetts Institute of Technology, 1982.

3 丘志杰, 罗蕾. 嵌入式Java反射机制的设计与实现. 计算机应用, 2010, 30(2): 398–401, 422.

4 张聪品, 刘超. 基于JAVA反射机制的规则引擎设计与实现. 河南师范大学学报: 自然科学版, 2010, 38(3): 36–39.

5 Stefano AD, Fargetta M, Tramontana E. Computational reflection for embedded java systems. Meersman R, Tari Z. On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops: Lecture Notes in Computer Science. Berlin Heidelberg, Germany, 2003, (2889): 437–450. [doi: 10.1007/b94345]

6 王万森, 龚文. Java动态类加载机制研究及应用. 计算机工程与设计, 2011, 32(6): 2154–2158.

7 Shams Z, Edwards SH. Reflection support: Java reflection made easy. The Open Software Engineering Journal, 2014, 7(1): 38–52.

8 邱万彬, 张国杰, 裘鸿林. Java中的组件复用相关技术. 计算机应用, 2005, 25(1): 73–75.

9 杨丰盛.Android应用开发揭秘.北京:机械工业出版社,2010.

10 Jeong S, Lee K, Hwang J, *et al*. AndroStep: Android storage performance analysis tool. Software Engineering 2013. Bonn, Germany. 2013. 327–340.

11 You JM, Park IK. Android storage access control for personal information security. The Journal of The Institute of Internet, Broadcasting and Communication, 2013, 13(6): 123–129. [doi: 10.7236/JIIBC.2013.13.6.123]

12 Kim H, Shin D. Cross-layered view on android storage IO system. 7th International Conference on Computing and Convergence Technology. Seoul, Korea. 2012. 1102–1105.

13 Lim SH, Lee S, Ahn WH. Applications IO profiling and analysis for smart devices. Journal of Systems Architecture, 2013, 59(9): 740–747. [doi: 10.1016/j.sysarc.2013.02.005]

14 Malony AD, Hammerslag DH, Jablonowski DJ. Traceview: A trace visualization tool. Zima HP. Parallel Computation: Lecture Notes in Computer Science. Berlin Heidelberg, Germany. 1991. 102–144.