2021年9月14日

名词解释

- 数字签名：
  - Android通过数字签名来**标识应用程序作者和在应用程序之间建立信任关系**，不是用来决定最终用户可以安装哪些应用程序。
  - Android系统不会安装运行任何一款未经数字签名的apk程序
- permission：
  - 通过Android官方市场，"打包安装器"安装应用程序时，所**显示的"权限"仅是在安装包内AndroidManifest.xml声明的值，而非应用程序实际上会调用的内容**。该值仅用来表明Android系统能向应用授予的最大可能的权限。即便一个"Hello World"式的应用程序，也可以在AndroidManifest.xml中声明所有可能的Android Permission。
  - Android有四种权限，其中signature和system是不对第三方APP提供的，只提供给系统自带软件；还有normal和dangerous，normal只要在AndroidManifest.xml中声明就可以使用，dangerous即使声明了也需要在使用时动态申请。
- Intent：
  - 相当于一个接口，在程序运行时激活连接不同的Activity、Service，传递数据。
  - 可以进行组件间的通信（一个应用的不同组件，不同应用的组件）
  - malware通过精心设计Intent Filter来截获未受到权限保护的隐式Intent。
- 污点分析：
  - 污点源，代表直接引入不受信任的数据或者机密数据到系统中

    污点汇聚点，代表直接产生安全敏感操作或者泄露隐私数据到外界

    无害处理，代表通过数据加密或者移除危害操作等手段使数据传播不再对软件系统的信息安全产生危害
  - 污点分析就是分析程序中由污点源引入的数据是否能够不经过无害处理，而直接传播到污点汇聚点。如果不能，说明信息流是安全的；否则，说明系统产生了隐私数据泄露或危险数据操作等问题 。
- 安卓中的四大组件：
  - Activity可以理解成是一个单独的屏幕，负责与用户进行交互，Activity之间要通过Intent进行通信。
  - Service进行背景任务，通常用于在后台处理耗时的操作或监控其他组件的运行状态。
  - Broadcast Receiver是一个全局监听器，接收特定广播消息并作出响应。
  - Content Provider的主要作用是跨程序共享数据，提供标准接口以支持多个应用存储和读取结构化数据。
- 组件间通信（ICC，inter-component communication）：
  - 主要通过Intent实现。
  - 组件要想接收到隐式的Intent，必须在manifest文件中声明Intent Filter。
  - 组件要想和其它应用的组件通信，其exported属性要为true。
- callback：一个函数作为另一个函数的参数时，这个函数就叫做callback，比如button.on('click', function(){})，这里的function()就是一个callback，应为该函数不是用户调用的，而是作为on函数的参数被调用的。
- recall：召回率，表示样本中正例有多少被预测正确了，就是预测的正类占了真实正类的多少，覆盖了多少的正类。
- 精确率是针对我们预测结果而言的，它表示的是预测为正的样本中有多少是真正的正样本

- Manifest.xml文件功能
  - 配置APP的permission
  - 确定需要的最小支持API level， 这个级别在build.gradle文件中也能定义，字段是minSdkVersion。在AndroidManifest.xml文件中定义的情况比较少。
  - 确认APP运行需要的硬件支持，比如相机、蓝牙等
  - 描述应用的各个组件，他需要定义组件的表现形式(组件名、主题、启动类型)，组件可以响应的操作等。
- ART(Android Runtime)：可以理解为Android运行环境，其前身是Dalvik。
- Android API：
  - API级别：每个Android版本对应一个API level， 让Android平台可以描述它支持的框架API的最高版本，让应用程序可以描述它需要的框架API版本，避免了一定的版本不兼容问题。
- IMEL(International Mobile Equipment Identity)，国际移动设备识别码，用于在移动电话网络中识别每一部独立的手机等移动通信设别，相当于移动设备的身份证。
- payload：往往指发挥关键作用的关键代码。
- F-measure（F值）：是一种统计量，又称为F-score，是Precision和recall的加权调和平均，用于评价分类模型的好坏。

AndroGuard

- 教程地址：Welcome to Androguard's documentation! — Androguard 3.4.0 documentation
- 启动方法：进入androguard目录下，使用*source venv-androguard/bin/activate*命令激活androguard。
- 分析apk命令：*androguard analyze*，会启动一个iPython shell便于后续的操作。
- 开始分析：*a, d, dx = AnalyzeAPK("examples/android/abcore/app-prod-debug.apk")*，其中a是apk对象，d是DalvikVMFormat对象，dx是Analysis对象。在APK对象中，可以得到所有与APK相关的信息，比如 package name, permissions, the AndroidManifest.xml or its resources。DalvikVMFormat对象对应的是从APK文件中提取出来的dex文件，可以从中获得calsses、methods、strings等信息；Analysis对象包含特殊的classes。
- APK信息获取方法：
  - a.get_permissions()：获取权限信息。
  - a.get_activities()：获取活动信息。
  - a.get_package()：获取package name。
  - a.get_apk_name()：获取app name。

    等一系列方法。
- decompile指令，可以用来产生方法的CFG，具体的可以使用androguard decompile --help查看具体参数。
- Androguard可以提供三种不同的字节码形式：原始字节(Raw bytes)、分解的表示(disssembled representation)、反汇编的表示(decompiled representation)。其中字节码以16位为单位构成。
- Session（会话）：用于保存当前的工作用于稍后返回。但是经验告诉我们不要使用Sessions进行批量分析，可以使用AndroAuto
- 除了使用DAD反汇编器，还可以使用JADX，不过需要另外下载。
- Androguard可以得到APK的签名证书。依靠：androguard.core.bytecodes.apk；此外，还可以验证apk签名，依靠apksighner。
- Androguard可以获得APK的manifest.xml文件，通过androguard axml，该指令也可以用来解码各种xml文件；也可以获得resources.arsc文件，通过androguard arsc；

- over-approximation：相当于可能性分析，需要产生的测试用例有很高的覆盖率，但中间可能有一些误报；而under-approximation，是要求产生的测试用例有很高的准确率，当然覆盖率就会低一些。
- event handler：事件句柄，指在事件发生时进行的动作。
- Smali是用于Dalvik的反汇编程序实现，汇编工具为smali.jar，一个Smali文件对应的是一个Java类，更准确说是一个.class文件。
  - 基础类型：V-void, Z-boolean, B-byte, S-short, C-char, I-int, J-long, F-float, D-double。
  - 对象：Objuct类型，在引用时，使用L开头，后面紧接着完整的包名，eg：java.lang.String对应的Smali语法法则是 Ljava/lang/String。
  - 数组：数组就是在左边加一个方括号，eg：[I 等同于Java的 int[]，每多一维就加一个方括号。
  - 方法的声明于调用：官方模板为：

    ```
    Lpackage/name/ObjectName;->MethodName(III)Z
    ```

    - `Lpackage/name/ObjectName;`用于声明具体的类型，以便VM寻
    - `MethodName(III)Z`，其中MethodName为具体的方法名，`()`中的字符，表示了参数数量和类型，即3个int型参数，Z为返回值的类型，即返回Boolean类型
  - Smali中，一个寄存器可以存储32位长度的类型，声明寄存器数量的方式为：`.registers N`
- Java注解(Annotation)，又称Java标注，是JDK5.0引入的一种注释机制，Java 语言中的类、方法、变量、参数和包等都可以被标注。

安卓

- 安卓6.0之前是安装时权限控制，6.0及之后是运行时权限控制，安卓6.0发布时间是2015年，这之前的论文中的权限机制应该都是安装时权限控制。
- Android使用Sandbox来进行远程隔离和数据隔离。
- 签名，保证应用不会被第三方篡改，证书在安装时被验证。
- Intent：应用传递消息的结构，可以启动activity、启动service、发出broadcast等。

  它有下面这五个属性：

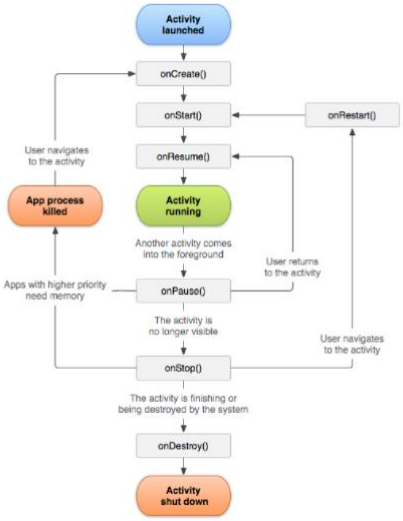
属性	说明	
ComponentName	启动的组件名称	explicit/ implicit
Action	执行的动作，系统定义或者应用自定义动作	
Data	传递的数据和数据类型	
Category	组件的类型	
Extras	额外的信息，key-value 对	
Flags	标志位	

  - Intent Filter：声明其接受的Intent类型（Implicit）。Action匹配，filter中某个action需要与Intent的action匹配，filter为空，无匹配，intent为空，filter至少有一个action，则匹配。Category匹配，intent为空，则匹配任意filter。Data匹配，按照Android下的每个data的链接进行匹配的。
  - PendingIntent用于包裹Intent，授权其他应用使用该Intent，不是立即执行的Intent，而是在未来指定的行为下才执行。

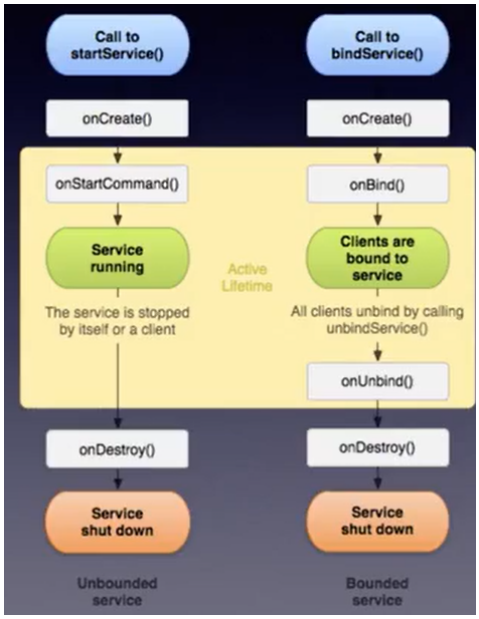
- Activity：声明时主要包含四个部分，名称、其他属性、Main activity（打开应用的第一个界面）、intent filter。
    - lifecycle：

      使用Bundle来存储lifecycle中不同状态切换的数据。

n.



    - activity之间的转换就严格按照上图进行。
    - 一个activity启动另一个activity有两种方法：startActivity和startActivityForResult（onActivityResult结果反馈）
- Tasks：是用户交互的Activity的集合，其中Activity类似于栈的排列方式，称为back stack。
- Recents Screen是用来查看历史的Activity和Task。
- Fragment作为一个Activity的子集或一部分，其具有单独的生命周期，独立接受输入事件，并且在activity运行时可以添加或者删除。可以认为其为子activity或者模块化。
- 进程间通信：
    - Intent的底层石油Bundle实现的。
    - Bundle是一个key-value形态。
    - 类的实例支持读取或存储为Parcel结构，需要实现Parcelable接口。
    - parcel用于包裹进程间传递的数据和结构。
- Service：
    - Scheduled服务：定时的服务
    - Started服务：其生命周期和启动它的组件无关。
    - Bound服务：一个组件bind服务，这个服务提供client-server形式的接口供该组件调用和交互，主要用于IPC，其生命周期和绑定的组件有关，如果没有组件绑定则销毁。主要函数：bindService、onBind
- IntentService继承与Service，使用一个工作线程来处理所有的服务启动请求。
- Bound Service：是c-s接口的服务器，其允许组件绑定该服务，并发送请求，接收响应以及其它IPC行为。
- 需要创建IBinder：创建Binder类，使用Messenger，使用ALDL。
- AIDL：Android接口定义语言，定义IPC的接口，需要自己处理多线程的问题，AIDL接口的定义：定义一个AIDL接口，实现接口方法，暴露接口实例到client。
- Service的lifecycle：



    - Foreground service：展示出来的Service，比如通知栏上的音乐小框，下载的进度条等。
- Broadcast Receiver：广播是Android系统与应用直接的消息传递方式，例如系统事件发生时或者应用事件发生。Broadcast Receiver注册用于接受指定类型的广播的组件。
- Content Provider：用于应用数据的访问，数据的存储和数据的共享。可用于IPC下的数据访问和管理。
- 应用自定义权限：目的在于暴露其数据、资源或组件访问给其它应用，通常市统一组织开发的多款应用之间的共享和访问或是不同组织开发的但用于共享其数据或者行为。
- 一个APP内不同的Activity、Service之间调用不用管权限，一个APP的Activity(Service)调用另一APP有权限的的Activity(Service)时必须有其权限才能调用。
- 可以指定一个权限，来限制可以接受的Broadcast Receiver。
- Android Support Library：给老版本的系统提供新的API的库，Android Support Library在命名上都有一个版本号（例如v7，v4），Android Support Library 可以分为两类：兼容库和组件库， 前者主要作用是解决向后兼容问题：来让APP运行在低版本手机上的时候能够使用新版本系统才有的特性。 需要保证工程的minSdkVersion API要大于Library支持的版本 。
- Compat类：名称以Compat结尾的类做同样的事，实现最新的API在旧版本Android上使用。

安卓开发

- Android项目中添加任何资源都会在R文件中生成一个相应的资源id。
- Android中设置main activity也是通过标签完成的，intent-filter也是在manifest文件中定义的。
- Android系统采用Toast来提供一种提醒方式，为了在程序中将一些短小的信息通知给用户，这些信息会在一段时间后自动消失，不会占用任何屏幕时间。
- 对于隐式的intent，只能指定一个action，但是可以指定多个category，只要有一个category匹配上了就算category匹配上了。隐式intent还可以有data标签。
- intent通过putExtra()方法在activity之间传递信息。
- startActivityForResult()方法也是用来启动活动的，但这个方法在活动销毁的时候能够返回一个结果给上一个活动。本质上还是通过intent来完成数据传递的。
- Android使用Task来管理活动，一个Task就是一组存放在栈里的活动的集合，这个栈也别称为返回栈（Back Stack）。系统总是回显示处于栈顶的活动给用户。

- 活动被回收之前一定会调用onSaveInstanceState()回调方法来保存临时数据，该方法会携带一个Bundle类型的参数，Bundle提供了一系列的方法用于保存数据，比如使用putString()、putString()分别保存字符串、整型。然后当被回收的活动再次被调用使用onCreate()时，可以选择一个Bundle类型参数来恢复数据。
- 碎片(Fragment)是一种可以嵌入在活动中的UI片段，它能让程序更加合理和充分地利用大屏幕的空间，特别适合也通常用于平板等大屏设备，其也有自己的生命周期和布局，甚至可以将其理解为一个迷你型的活动。
- 关于广播，发送广播需要Intent，接收广播则靠广播接收器（Broadcast Receiver）。
    - 标准广播：一种完全异步执行的广播，在广播发出之后，所有的广播接收器几乎同时收到这条广播消息，没有先后顺序而言，也无法被截断。
    - 有序广播：一种同步执行的广播，在广播发出后，同一时刻只有一个广播接收器能够收到这条广播消息，优先级高的广播器先收到广播，也能够被截断。
- 广播有动态和静态：动态注册的广播可以自由地控制注册与注销，但是必须要程序启动后才能接收到广播（因为注册的逻辑写在onCreate()方法中）；静态的广播接收器需要在Manifest中注册才行，比如开机自启动app。
- Android提供了本地广播机制，使用这个机制发出的广播只能够在应用程序内部进行传递，并且广播接收器也只能接受来自本地应用程序发出的广播，很大程度上提高了安全性。
- Android系统内置了一个轻量级得关系型数据库SQLite，运算速度极快，占用资源很少。SQLite不仅支持SQL语法，还遵勖数据库的ACID事务。LitePal是一个好用的帮助操作SQLite数据库的开源库。

- 运行时权限：在软件使用到某一危险权限的时候再去向用户申请，避免了一个无关紧要的权限导致不能够使用整个app的情况。
- 每个危险权限都属于一个权限组，当我们在运行时授权了一个权限，那么该权限对应的权限组的其它权限也会同时被授权。

- 内容提供商（Content Provider）主要用于在不同的应用程序之间实现数据共享功能，它提供一套完整的机制，允许一个程序访问另一个程序的数据，同时还能保证被访问数据的安全性。
- 内容URI给内容提供器中的数据建立了唯一标识符，它主要由两部分组成，权限(authority)和路径(path)，权限是用于对不同的应用程序做区分的，一般为了避免冲突都会采用程序包名的方式进行命名；路径则是用于对同一应用程序中不同的表做区分的，通常会添加到权限后面。内容URI最标准的格式写法如下：

```
content://com.example.app.provider/table1
content://com.example.app.provider/table2
```

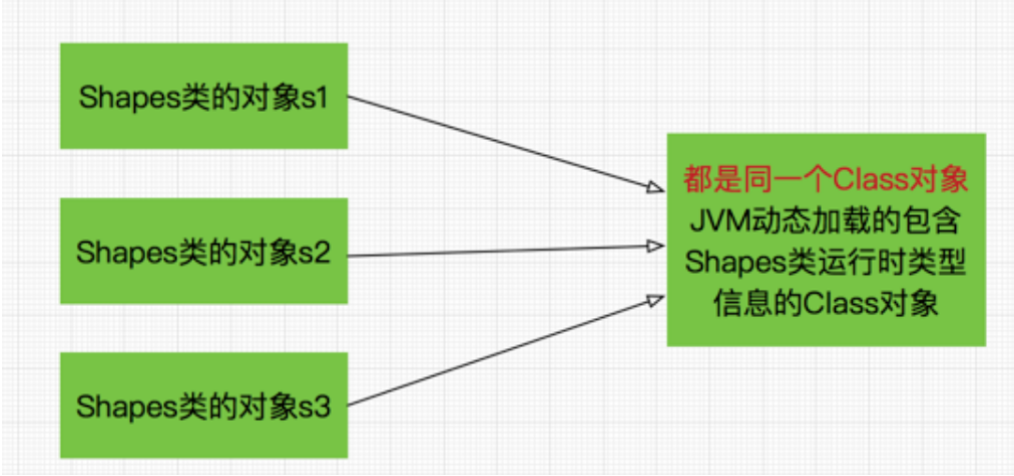
- 如果想要访问内容提供器中共享的数据，就一定要借助ContentResolver类。想要实现跨程序共享数据的功能，官方推荐的就是使用内容提供器，可以通过继承ContentProvider类的方式来创建一个自己的内容提供器，因为所有的CRUD操作都一定要匹配到相应的内容URI格式才能进行的，解决了安全问题。

- 通知(Notification)： 当某个应用程序希望向 用户发出一些提示信息，而该应用程序又不在前台运行时，就可以借助通知来实现。发出一 条通知后，手机最上方的状态栏中会显示一个通知的图标，下拉状态栏后可以看到通知的详细内容 。
- 读写SD卡为危险权限，而使用应用关联目录就可以跳过读写SD卡的步骤。

- WebView控件可以帮助我们在自己的应用程序里嵌入一个浏览器，从而可以轻松展示各种各样的网页。
- 服务（Service）是Android中实现程序后台运行的解决方案，非常适合去执行那些不需要和用户交互而且还要求长期运行的任务。服务并不是运行在一个独立的进程中的，而是依赖于创建服务时所在的应用程序进程，当某个应用程序被杀死时，所有依赖于该进程的服务也会停止运行。每个服务也需要在Manifest文件中注册才能生效。启动和停止也是主要借助Intent完成。
- Android中的异步消息处理主要有四部分组成：Message、Handler、MessageQueue和Looper。
    - Message：是在线程之间传递消息，它可以在内部携带少量的信息，用于在不同线程之间交换数据。
    - Handler：相当于"处理者"，主要是用于发送和处理消息，发送消息一般使用Handler中的sendMessage()方法，而发送的消息经过一系列处理后，最终会传递到Handler的handleMessage()方法中。
    - MessageQueue：消息队列，主要用于存放所有通过Handler发送的消息，等待被处理，每个线程中只会有一个MessageQueue对象。
    - Looper：是每个线程中的MessageQueue的管家，调用Looper的loop()方法后，就会进入到一个无限循环当中，然后每当发现MessageQueue中存在一条消息，就会将他取出，并传递到Handler的handleMessage()方法中，每个线程也只有一个Looper对象。
- Intent传递对象通常有两种实现方式：Serializable和Parcelable。
    - Serializable是序列化的意思，表示将对象转换成可传输的状态，序列化后的对象可以在网络上进行传输，也可以存储在本地。
    - Parcelable方式的实现原理是将一个完整的对象进行分解，分解后的每一部分都是Intent所支持的数据类型。
- 关于反汇编得到的文件的解释：
    - 顶层文件夹Support：指的应该是android提供的很多Support Library(支持库)。
    - Support下文件夹：也是 Android Support Library包含的依赖包。
        - fragment：一个专门解决Android碎片化的类， 通过它可以让同一个程序适配不同的屏幕 。

Java

- RTTI（Run-Time Type Identification）运行时类型识别 ，作用是在运行时识别一个对象的类型和类的信息；这里分两种：传统的"RRTI",它假定我们在编译期已知道了所有类型，另外一种是反射机制，它允许我们在运行时发现和使用类型的信息 ，在Java中用来表示运行时类型信息的对应类就是Class类，Class类也是一个实实在在的类，存在于JDK的java.lang包中。Java中每个类都有一个Class对象，



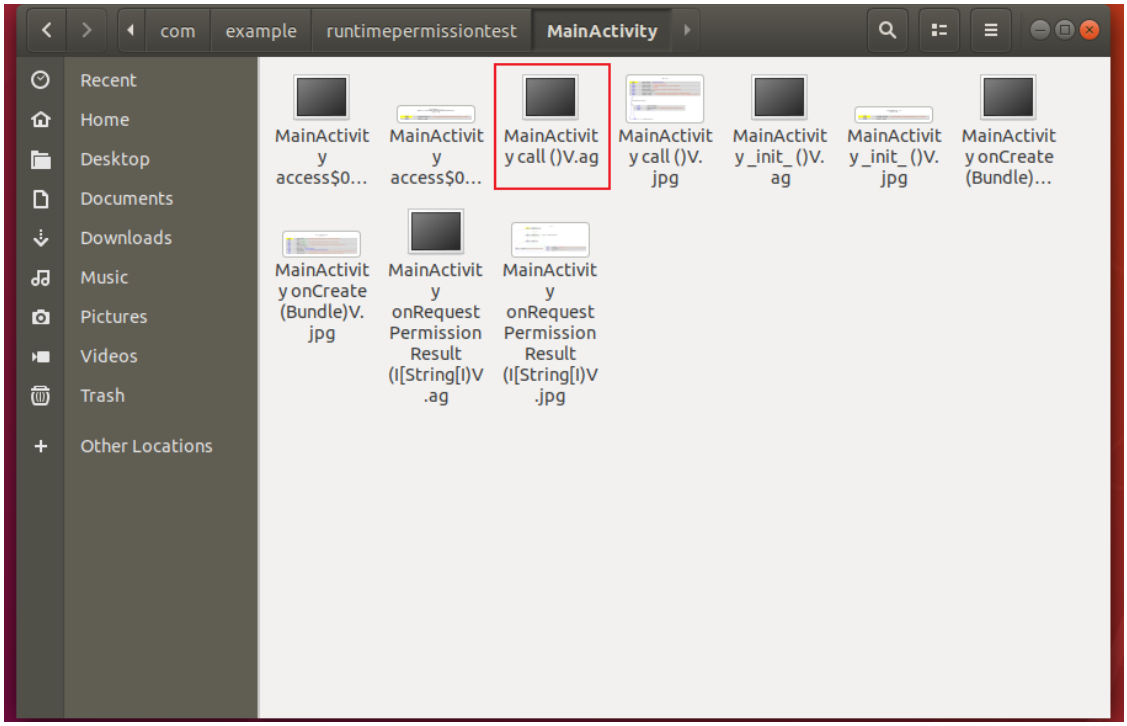
- 反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性，这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。
- Annotation是一个辅助类，具有让编译器进行编译检查的作用，也能方便我们了解程序的大致结构。
- final关键字：修饰类，表示这个类不能够被继承，其中的所有方法也隐式地被指定为final方法；修饰方法，明确禁止该方法在子类中被覆盖；修饰变量，是final关键字用的最多的地方，表示该变量在初始化后不能再改变其值。

问题

- androguard使用decompile时间消耗十分长，即使是十分简单的APP。

  到jpg是分成两部的，生成cfg过程是比较慢的，下来试下生成反汇编diama应该快一些

- .ag结尾的文件是什么，androguard的缩写？有必要了解吗？



- XREFs are generated for Classes, Methods, Fields and Strings.

  XREFs--crossreferences

  我理解的就是方法、类之间的相互引用，就是普通的调用关系。

  语言方面，系统内部，比较底层的概念

```
In [10]: for meth in dx.classes['Ltests/androguard/TestActivity;'].get_methods():
    ...:     print("inside method {}".format(meth.name))
    ...:     for _, call, _ in meth.get_xref_to():
    ...:         print("  calling -> {} -- {}".format(call.class_name, call.name))
    ...:
inside method testCall1
  calling -> Ljava/lang/StringBuilder; -- toString
  calling -> Ljava/lang/StringBuilder; -- append
  calling -> Ljava/lang/StringBuilder; -- <init>
  calling -> Ljava/io/PrintStream; -- println
inside method testCalls
  calling -> Ljava/lang/Object; -- getClass
  calling -> Ljava/io/PrintStream; -- println
  calling -> Ltests/androguard/TestIfs; -- testIF
  calling -> Ltests/androguard/TestActivity; -- testCall2
[...]
```