lostinai的专栏

用勇气改变可以改变的事情 用胸怀接受不能改变的事情 在选择积极态度的同时 须保持平和的心态

个人资料

lostinai

访问： 467490次
积分： 6614
等级：
排名： 第2986名

原创： 67篇 转载： 793篇
译文： 0篇 评论： 20条

文章搜索

文章分类

Smali语法详解

2015-10-08 17:48 2751人阅读 评论(0) 收藏 举报

分类： Android安全（41） 反编译（8）

目录(?) [+]

smali文件格式

每个smali文件都由若干条语句组成，所有的语句都遵循着一套语法规则。在smali 文件的头3 行描述了当前类的一些信息，格式如下：

```
[java]

.class < 访问权限> [ 修饰关键字] < 类名>
.super < 父类名>
.source <源文件名>
```

打开MainActivity.smali 文件，头3 行代码如下：

```
[java]

.class public Lcom/droider/crackme0502/MainActivity;     //指令指定了当前类的类名。
.super Landroid/app/Activity;<span style="white-space:pre">             </span>//指令指定了当前类的父类。
.source "MainActivity.java"<span style="white-space:pre">             </span>//指令指定了当前类的源文件名。
```

smali文件中字段的声明使用“.field”指令。字段有静态字段与实例字段两种。静态字段的声明格式如下:

```
[java]

# static fields
.field < 访问权限> static [ 修饰关键字] < 字段名>:< 字段类型>
```

实例字段的声明与静态字段类似，只是少了static关键字，它的格式如下:

```
[java]

# instance fields
.field < 访问权限> [ 修饰关键字] < 字段名>:< 字段类型>
```

```
[java]

比如以下的实例字段声明。
# instance fields   //baksmali 生成的注释
.field private btnAnno:Landroid/widget/Button;  //私有字段
```

smali 文件中方法的声明使用“.method ”指令,方法有直接方法与虚方法两种。
直接方法的声明格式如下:

Activity (18)
String (8)
源码研究 (33)
开发工具 (14)
Android动画 (16)
多媒体 (3)
Xml (12)
硬件相关 (2)
ViewFlipper与ViewPager (11)
数据结构和算法 (8)
View (33)
英语学习 (2)
SQL (3)
JAVA中的线程 (8)
IOS (2)
C/C++ (49)
编译原理 (1)
WebService (4)
Linux (4)
JVM (12)
校园招聘 (13)
反编译 (9)
Leetcode (1)
Fragment (2)
Cocos2d-x (15)
游戏服务器 (1)
计算机体系结构 (0)
hadoop学习 (0)
NDK (12)
开源控件 (39)
应用开发框架 (28)
HAL (1)
文章积累 (5)
Python (4)
网络安全 (4)
Android安全 (42)
机器学习 (24)
Binder机制 (1)
ReactNative (1)

文章存档

2017年03月 (2)
2017年02月 (2)
2017年01月 (4)
2016年11月 (1)
2016年10月 (7)

展开

阅读排行

仿新浪微博的ListView下: (3569)
Android 视频缩略图之Me (3387)
使用ViewPager实现高仿 (3337)
高仿launcher和墨迹左右 (3310)
C/C++笔试题（很多） (3302)
Android笔面试题 (3295)
ListView所扩展的各种牛: (3217)
九月十月百度人搜，阿里 (3066)
Android GridView属性 (3065)
android ListView异步加载 (2824)

```java
# direct methods                    //添加的注释
.method <访问权限> [ 修饰关键字] < 方法原型>
    <.locals>                   //指定了使用的局部变量的个数
 [.parameter]                     //指定了方法的参数
 [.prologue]                      //指定了代码的开始处，混淆过的代码可能去掉了该指令
 [.line]                        //指定了该处指令在源代码中的行号
<代码体>
.end method
```

虚方法的声明与直接方法相同，只是起始处的注释为“virtual methods”,如果一个类实现了接口，会在smali 文件中使用“.implements ”指令指出,相应的格式声明如下:

```java
# interfaces
.implements < 接口名>         //接口关键字

如果一个类使用了注解，会在 smali 文件中使用“.annotation ”指令指出,注解的格式声明如下:
# annotations
.annotation [ 注解属性] < 注解类名>
    [ 注解字段 =  值]
.end annotation
```

注解的作用范围可以是类、方法或字段。如果注解的作用范围是类，“.annotation ”指令会直接定义在smali 文件中，如果是方法或字段，“.annotation ”指令则会包含在方法或字段定义中。例如:

```java
# instance fields
.field public sayWhat:Ljava/lang/String;            //String 类型 它使用了 com.droider.anno.MyAnnoField 注解，注解字段info 值为“Hello my friend”
    .annotation runtime Lcom/droider/anno/MyAnnoField;
        info = "Hello my friend"
    .end annotation
.end field
```

Android 程序中的类

1、内部类

Java 语言允许在一个类的内部定义另一个类，这种在类中定义的类被称为内部类（Inner Class）。内部类可分为成员内部类、静态嵌套类、方法内部类、匿名内部类。在反编译dex 文件的时候，会为每个类单独生成了一个 smali 文件，内部类作为一个独立的类，它也拥有自己独立的smali 文件，只是内部类的文件名形式为“[外部类]$[内部类].smali ”，例如：

```java
class Outer {
   class Inner{}
}
```

反编译上述代码后会生成两个文件：Outer.smali 与Outer$Inner.smali。打开文件，代码结构如下：

```java
.class public Lcom/droider/crackme0502/MainActivity$SNChecker;
.super Ljava/lang/Object;
.source "MainActivity.java"

# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/droider/crackme0502/MainActivity;
.end annotation
.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x1
    name = "SNChecker"
.end annotation
```

```
# instance fields
.field private sn:Ljava/lang/String;
.field final synthetic this$0:Lcom/droider/crackme0502/MainActivity;

# direct methods
.method public constructor
<init>(Lcom/droider/crackme0502/MainActivity;Ljava/lang/String;)V
……
.end method

# virtual methods
.method public isRegistered()Z
……
.end method
```

发现它有两个注解定义块“Ldalvik/annotation/EnclosingClass;”与“Ldalvik/annotation/ InnerClass; ”、两个实例字段 sn 与this$0 、一个直接方法 init()、一个虚方法isRegistered() 。this$0 是内部类自动保留的一个指向所在外部类的引用。左边的 this 表示为父类的引用，右边的数值0 表示引用的层数。

2、监听器

Android程序开发中大量使用到了监听器，如Button的点击事件响应OnClickListener、Button的长按事件响应OnLongClickListener、ListView列表项的点击事件响应 OnItemSelected-Listener等。

实例源码以及反编译设置按钮点击事件监听器的代码如下：

```
[java]

public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnAnno = (Button) findViewById(R.id.btn_annotation);
        btnCheckSN = (Button) findViewById(R.id.btn_checksn);
        edtSN = (EditText) findViewById(R.id.edt_sn);

        btnAnno.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                getAnnotations();
            }
        });

        btnCheckSN.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                SNChecker checker = new SNChecker(edtSN.getText().toString());
                String str = checker.isRegistered() ? "注册码正确" : "注册码错误";
                Toast.makeText(MainActivity.this, str, Toast.LENGTH_SHORT).show();
            }
        });

    }
```

```
[java]

反编译如下：
.method public onCreate(Landroid/os/Bundle;)V
     .locals 2
    .parameter "savedInstanceState"
    ……
    .line 32
    iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;->btnAnno:
    Landroid/widget/Button;
    new-instance v1, Lcom/droider/crackme0502/MainActivity$1; #新建一个
    MainActivity$1实例
    invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$1;
->&lt;init&gt;(Lcom/droider/crackme0502/MainActivity;)V # 初始化MainActivity$1
实例
```

```
    invoke-virtual {v0, v1}, Landroid/widget/Button;
->setOnClickListener(Landroid/view/View$OnClickListener;)V # 设置按钮点击事件
监听器
    .line 40
    iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;
->btnCheckSN:Landroid/widget/Button;
    new-instance v1, Lcom/droider/crackme0502/MainActivity$2; #新建一个
MainActivity$2实例
    invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$2
-><init>(Lcom/droider/crackme0502/MainActivity;)V; # 初始化MainActivity$2实例
    invoke-virtual {v0, v1}, Landroid/widget/Button;
->setOnClickListener(Landroid/view/View$OnClickListener;)V#设置按钮点击事件
监听器
    .line 50
    return-void
.end method
```

在MainActivity$1.smali 文件的开头使用了".implements "指令指定该类实现了按钮点击事件的监听器接口，因此，这个类实现了它的OnClick()方法，这也是我们在分析程序时关心的地方。另外，程序中的注解与监听器的构造函数都是编译器为我们自己生成的，实际分析过程中不必关心。

3、注解类

注解是Java 的语言特性，在 Android的开发过程中也得到了广泛的使用。Android系统中涉及到注解的包共有两个：一个是dalvik.annotation；另一个是 android.annotation。

例如：

```
[java]

# annotations
.annotation system Ldalvik/annotation/AnnotationDefault;
    value = .subannotation Lcom/droider/anno/MyAnnoClass;
        value = "MyAnnoClass"
    .end subannotation
.end annotation
```

除了SuppressLint与TargetApi注解，android.annotation 包还提供了SdkConstant与Widget两个注解，这两个注解在注释中被标记为"@hide"，即在 SDK 中是不可见的。SdkConstant注解指定了SDK中可以被导出的常量字段值，Widget 注解指定了哪些类是 UI类，这两个注解在分析Android程序时基本上碰不到，此处就不去探究了。

4、自动生成的类

使用 Android SDK 默认生成的工程会自动添加一些类。例如：

```
[java]

public final class R {
    public static final class attr {      //属性
    }
    public static final class dimen {      //尺寸
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {    //图片
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {        //id标识
        public static final int btn_annotation=0x7f080000;
        public static final int btn_checksn=0x7f080002;
        public static final int edt_sn=0x7f080001;
        public static final int menu_settings=0x7f080003;
    }
```

```
    public static final class layout {    // 布局
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {    // 菜单
        public static final int activity_main=0x7f070000;
    }
    public static final class string {    // 字符串
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050001;
        public static final int menu_settings=0x7f050002;
        public static final int title_activity_main=0x7f050003;
    }
    public static final class style {    // 样式
        public static final int AppTheme=0x7f060000;
    }
}
```

由于这些资源类都是R 类的内部类，因此它们都会独立生成一个类文件，在反编译出的代码中，可以发现有R.smali、R$attr.smali 、R$dimen.smali、R$drawable.smali、R$id.smali、R$layout.smali、R$menu.smali 、R$string.smali 、R$style.smali 等几个文件。

阅读smali反编译的代码

smali 文件中的语句特点：

1、循环语句

在 Android开发过程中，常见的循环结构有迭代器循环、for 循环、while循环、do while 循环。我们在编写迭代器循环代码时，一般是如下形式的代码：

```
[java]

Iterator< 对象> <对象名> = <方法返回一个对象列表>;
for (< 对象> <对象名> : <对象列表>) {
[处理单个对象的代码体]
}
或者：
Iterator< 对象> <迭代器> = <方法返回一个迭代器>;
while (<迭代器>.hasNext()) {
 <对象> <对象名> = <迭代器>.next();
[处理单个对象的代码体]
}
```

```
[java]

.method private iterator()V
    .locals 7
    .prologue
    .line 34
    const-string v4, "activity"
    invoke-virtual {p0, v4}, Lcom/droider/circulate/MainActivity;->
    getSystemService
         (Ljava/lang/String;)Ljava/lang/Object;  # 获取ActivityManager
    move-result-object v0
    check-cast v0, Landroid/app/ActivityManager;
    .line 35
    .local v0, activityManager:Landroid/app/ActivityManager;
    invoke-virtual {v0}, Landroid/app/ActivityManager;->getRunningAppProcesses()
    Ljava/util/List;
    move-result-object v2    #正在运行的进程列表
    .line 36
    .local v2, psInfos:Ljava/util/List;,
    "Ljava/util/List<Landroid/app/ActivityManager$RunningAppProcessInfo;>;"
    new-instance v3, Ljava/lang/StringBuilder;  # 新建一个StringBuilder 对象
    invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V  # 调用
    StringBuilder 构造函数
    .line 37
    .local v3, sb:Ljava/lang/StringBuilder;
    invoke-interface {v2}, Ljava/util/List;->iterator()Ljava/util/Iterator;
 #获取进程列表的迭代器
    move-result-object v4
    :goto_0 #迭代循环开始
    invoke-interface {v4}, Ljava/util/Iterator;->hasNext()Z #开始迭代
    move-result v5
```

```
    if-nez v5, :cond_0  # 如果迭代器不为空就跳走
    .line 40
    invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
    String;
    move-result-object v4  # StringBuilder转为字符串
    const/4 v5, 0x0
    invoke-static {p0, v4, v5}, Landroid/widget/Toast;->makeText
         (Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/
         widget/Toast;
    move-result-object v4
    invoke-virtual {v4}, Landroid/widget/Toast;->show()V # 弹出StringBuilder
    的内容
    .line 41
    return-void  # 方法返回
    .line 37
    :cond_0
    invoke-interface {v4}, Ljava/util/Iterator;->next()Ljava/lang/Object;
    # 循环获取每一项
    move-result-object v1
    check-cast v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
    .line 38
    .local v1, info:Landroid/app/ActivityManager$RunningAppProcessInfo;
    new-instance v5, Ljava/lang/StringBuilder;  # 新建一个临时的StringBuilder
    iget-object v6, v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
        ->processName:Ljava/lang/String;    #获取进程的进程名
    invoke-static {v6}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
    Ljava/lang/String;
    move-result-object v6
    invoke-direct {v5, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
    String;)V
    const/16 v6, 0xa #换行符
    invoke-virtual {v5, v6}, Ljava/lang/StringBuilder;->append(C)Ljava/
    lang/StringBuilder;
    move-result-object v5 # 组合进程名与换行符
    invoke-virtual {v5}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
    String;
    move-result-object v5
    invoke-virtual {v3, v5}, Ljava/lang/StringBuilder; # 将组合后的字符串添加到
    StringBuilder 末尾
        ->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    goto :goto_0  #跳转到循环开始处
.end method
```

这段代码的功能是获取正在运行的进程列表，然后使用Toast弹出所有的进程名。

forCirculate() 方法如下：

```
[java]

.method private forCirculate()V
    .locals 8
    .prologue
    .line 47
    invoke-virtual {p0}, Lcom/droider/circulate/MainActivity;-
        >getApplicationContext()Landroid/content/Context;
    move-result-object v6
    invoke-virtual {v6}, Landroid/content/Context;    #获取PackageManager
        ->getPackageManager()Landroid/content/pm/PackageManager;
    move-result-object v3
    .line 49
    .local v3, pm:Landroid/content/pm/PackageManager;
    const/16 v6, 0x2000
    .line 48
    invoke-virtual {v3, v6}, Landroid/content/pm/PackageManager;
        ->getInstalledApplications(I)Ljava/util/List;  #获取已安装的程序列表
    move-result-object v0
    .line 50
    .local v0, appInfos:Ljava/util/List;,"Ljava/util/List<Landroid/content/pm
    /ApplicationInfo;>;"
    invoke-interface {v0}, Ljava/util/List;->size()I  # 获取列表中ApplicationInfo
    对象的个数
    move-result v5
    .line 51
    .local v5, size:I
    new-instance v4, Ljava/lang/StringBuilder;          # 新建一个
    StringBuilder 对象
    invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V  # 调用
    StringBuilder 的构造函数
```

```
    .line 52
    .local v4, sb:Ljava/lang/StringBuilder;
    const/4 v1, 0x0

    .local v1, i:I #初始化v1为0
    :goto_0 #循环开始
    if-lt v1, v5, :cond_0   #如果v1小于v5，则跳转到cond_0 标号处
    .line 56
    invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/
    lang/String;
    move-result-object v6
    const/4 v7, 0x0
    invoke-static {p0, v6, v7}, Landroid/widget/Toast; #构造Toast
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v6
    invoke-virtual {v6}, Landroid/widget/Toast;->show()V #显示已安装的程序列表
    .line 57
    return-void  # 方法返回
    .line 53
    :cond_0
    invoke-interface {v0, v1}, Ljava/util/List;->get(I)Ljava/lang/Object;
    # 单个ApplicationInfo
    move-result-object v2
    check-cast v2, Landroid/content/pm/ApplicationInfo;
    .line 54
    .local v2, info:Landroid/content/pm/ApplicationInfo;
    new-instance v6, Ljava/lang/StringBuilder;  # 新建一个临时StringBuilder对象
    iget-object v7, v2, Landroid/content/pm/ApplicationInfo;->packageName:
    Ljava/lang/String;
    invoke-static {v7}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
    Ljava/lang/String;
    move-result-object v7  # 包名
    invoke-direct {v6, v7}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
    String;)V
    const/16 v7, 0xa #换行符
    invoke-virtual {v6, v7}, Ljava/lang/StringBuilder;->append(C)Ljava/
    lang/StringBuilder;
    move-result-object v6  # 组合包名与换行符
    invoke-virtual {v6}, Ljava/lang/StringBuilder;->toString()Ljava/lang
    /String; #转换为字符串
    move-result-object v6
    invoke-virtual {v4, v6}, Ljava/lang/StringBuilder;-
        >append(Ljava/lang/String;)Ljava/lang/StringBuilder;  # 添加到循环外        的
StringBuilder 中
    .line 52
    add-int/lit8 v1, v1, 0x1 #下一个索引
    goto :goto_0 #跳转到循环起始处
.end method
```

这段代码的功能是获取所有安装的程序，然后使用Toast弹出所有的软件包名。

2、switch分支语句

packedSwitch()方法的代码如下：

```
[java]

.method private packedSwitch(I)Ljava/lang/String;
    .locals 1
    .parameter "i"
    .prologue
    .line 21
    const/4 v0, 0x0
    .line 22
    .local v0, str:Ljava/lang/String;  #v0为字符串，0表示null
    packed-switch p1, :pswitch_data_0  #packed-switch分支，pswitch_data_0指
    定case区域
    .line 36
    const-string v0, "she is a person"  #default分支
    .line 39
    :goto_0      #所有case的出口
    return-object v0 #返回字符串v0
    .line 24
    :pswitch_0    #case 0
```

```
    const-string v0, "she is a baby"
    .line 25
    goto :goto_0  #跳转到goto_0标号处
    .line 27
    :pswitch_1    #case 1
    const-string v0, "she is a girl"
    .line 28
    goto :goto_0  #跳转到goto_0标号处
    .line 30
    :pswitch_2    #case 2
    const-string v0, "she is a woman"
    .line 31
    goto :goto_0  #跳转到goto_0标号处
    .line 33
    :pswitch_3    #case 3
    const-string v0, "she is an obasan"
    .line 34
    goto :goto_0  #跳转到goto_0标号处
    .line 22
    nop
    :pswitch_data_0
    .packed-switch 0x0    #case  区域，从0开始，依次递增
        :pswitch_0  #case 0
        :pswitch_1  #case 1
        :pswitch_2  #case 2
        :pswitch_3  #case 3
    .end packed-switch
.end method
```

代码中的switch 分支使用的是 packed-switch 指令。p1为传递进来的 int 类型的数值，pswitch_data_0 为case 区域，在 case 区域中，第一条指令“.packed-switch”指定了比较的初始值为0 ，pswitch_0~ pswitch_3分别是比较结果为“case 0 ”到“case 3 ”时要跳转到的地址。

3、try/catch 语句

tryCatch()方法代码如下：

```
[java]

.method private tryCatch(ILjava/lang/String;)V
    .locals 10
    .parameter "drumsticks"
    .parameter "peple"
    .prologue
    const/4 v9, 0x0
    .line 19
    :try_start_0  # 第1个try开始
    invoke-static {p2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
 #将第2个参数转换为int 型
    :try_end_0    # 第1个try结束
    .catch Ljava/lang/NumberFormatException; {:try_start_0 .. :try_end_0} :
    catch_1 # catch_1
    move-result v1  #如果出现异常这里不会执行，会跳转到catch_1标号处
    .line 21
    .local v1, i:I    #.local声明的变量作用域在.local声明与.end local 之间
    :try_start_1  #第2个try 开始
    div-int v2, p1, v1  # 第1个参数除以第2个参数
    .line 22
    .local v2, m:I    #m为商
    mul-int v5, v2, v1  #m * i
    sub-int v3, p1, v5  #v3 为余数
    .line 23
    .local v3, n:I
    const-string v5, "\u5171\u6709%d\u53ea\u9e21\u817f\uff0c%d
        \u4e2a\u4eba\u5e73\u5206\uff0c\u6bcf\u4eba\u53ef\u5206\u5f97%d
        \u53ea\uff0c\u8fd8\u5269\u4e0b%d\u53ea"   # 格式化字符串
    const/4 v6, 0x4
    new-array v6, v6, [Ljava/lang/Object;
    const/4 v7, 0x0
    .line 24
    invoke-static {p1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
    aput-object v8, v6, v7
    const/4 v7, 0x1
    invoke-static {v1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
```

```
    aput-object v8, v6, v7
    const/4 v7, 0x2
    invoke-static {v2}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
    aput-object v8, v6, v7
    const/4 v7, 0x3
    invoke-static {v3}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8

    aput-object v8, v6, v7
    .line 23
    invoke-static {v5, v6}, Ljava/lang/String;
        ->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;
    move-result-object v4
    .line 25
    .local v4, str:Ljava/lang/String;
    const/4 v5, 0x0
    invoke-static {p0, v4, v5}, Landroid/widget/Toast;
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v5
    invoke-virtual {v5}, Landroid/widget/Toast;->show()V  # 使用Toast 显示格
    式化后的结果
    :try_end_1  #第2个try 结束
    .catch Ljava/lang/ArithmeticException; {:try_start_1 .. :try_end_1} :
    catch_0   # catch_0
    .catch Ljava/lang/NumberFormatException; {:try_start_1 .. :try_end_1} :
    catch_1   # catch_1
    .line 33
    .end local v1           #i:I
    .end local v2           #m:I
    .end local v3           #n:I
    .end local v4           #str:Ljava/lang/String;
    :goto_0
    return-void  # 方法返回
    .line 26
    .restart local v1       #i:I
    :catch_0
    move-exception v0
    .line 27
    .local v0, e:Ljava/lang/ArithmeticException;
    :try_start_2  #第3个try 开始
    const-string v5, "\u4eba\u6570\u4e0d\u80fd\u4e3a0" #“人数不能为0”
    const/4 v6, 0x0
    invoke-static {p0, v5, v6}, Landroid/widget/Toast;
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v5
    invoke-virtual {v5}, Landroid/widget/Toast;->show()V  # 使用Toast 显示异     常原因
    :try_end_2    #第3个try 结束
    .catch Ljava/lang/NumberFormatException; {:try_start_2 .. :try_end_2} :
    catch_1
    goto :goto_0 #返回
    .line 29
    .end local v0           #e:Ljava/lang/ArithmeticException;
    .end local v1           #i:I
    :catch_1
    move-exception v0
    .line 30
    .local v0, e:Ljava/lang/NumberFormatException;
    const-string v5, "\u65e0\u6548\u7684\u6570\u503c\u5b57\u7b26\u4e32"
    #“无效的数值字符串”
    invoke-static {p0, v5, v9}, Landroid/widget/Toast;
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v5
    invoke-virtual {v5}, Landroid/widget/Toast;->show()V  # 使用Toast 显示异
    常原因
    goto :goto_0 #返回
.end method
```

整段代码的功能比较简单，输入鸡腿数与人数，然后使用Toast弹出鸡腿的分配方案。传入人数时为了演示Try/Catch效果，使用了String 类型。代码中有两种情况下会发生异常：第一种是将String 类型转换成 int 类型时可能会发生 NumberFormatException异常；第二种是计算分配方法时除数为零的ArithmeticException异常。

在Dalvik 指令集中，并没有与Try/Catch相关的指令，在处理Try/Catch语句时，是通过相关的数据结构来保存异常

信息的。

小结

静态分析是软件分析过程中最基础也是最重要的一种手段，我们向Android逆向破解又迈进了一大步。

上一篇 Android开发资源获取国内代理（转载）
下一篇 [以早期版本为例]快速Dump爱加密的方法

我的同类文章

Android安全（41） 反编译（8）

- APK ROM 签名原理 2017-01-23 阅读 47
- Android安全攻防战，反编译… 2016-06-03 阅读 408
- Android中的Apk的加固(加壳… 2016-03-15 阅读 983
- 途牛原创|途牛Android App… 2016-01-11 阅读 432
- Android 安全学习 2015-11-08 阅读 409
- 研究Xposed相关一：Xpose… 2015-11-04 阅读 644
- Android中签名原理和安全性… 2017-01-23 阅读 74
- Android动态加载技术 简单… 2016-04-28 阅读 781
- 携程Android App插件化和动… 2016-03-10 阅读 199
- android Apk打包过程概述_a… 2015-11-24 阅读 174
- 研究Xposed相关二：如何ro… 2015-11-04 阅读 219

更多文章

参考知识库

猜你在找