

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

目录视图 摘要 订阅

个人资料



lostinai



访问：467490次

积分：6614

等级：BLOG 5

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

Android基础积累篇 (106)

Java基础积累篇 (43)

Android面试 (30)

ContentProvider (5)

ListView (52)

网络 (58)

人生哲理 (15)

设计模式 (13)

图片处理 (19)

SQLite (9)

项目源码 (15)

ubuntu (4)

BroadcastReceiver (9)

性能优化 (24)

内存溢出 (7)

数据存储 (6)

Service (15)

正则表达式 (8)

ExpandableListView (6)

多线程 (10)

Android布局 (38)

Json (6)

Handler (17)

Android线程 (24)

程序员3月书讯

【面试秘籍】开发者应该如何谈薪资

Python数据分析与机器学习

博客搬家，有礼相送

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

打开MainActivity.smali 文件，头3 行代码如下：

```
[java]
01. .class public Lcom/droider/crackme0502/MainActivity; //指令指定了当前类的类名。
02. .super Landroid/app/Activity; <span style="white-space:pre"> //指令指定了当前类的父类。
03. .source "MainActivity.java"<span style="white-space:pre"> //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]
01. # static fields
02. .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]
01. # instance fields
02. .field < 访问权限> [ 修饰关键字] < 字段名>:< 字段类型>

[java]
01. 比如以下的实例字段声明。
02. # instance fields //baksmali 生成的注释
03. .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)
- ViewPager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]

```
01. # direct methods           //添加的注释
02. .method <访问权限> [ 修饰关键字] < 方法原型>
03.     <.locals>               //指定了使用的局部变量的个数
04.     [.parameter]            //指定了方法的参数
05.     [.prologue]             //指定了代码的开始处，混淆过的代码可能去掉了该指令
06.     [.line]                 //指定了该处指令在源代码中的行号
07. <代码体>
08. .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]

```
01. # interfaces
02. .implements < 接口名>       //接口关键字
03.
04. 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：
05. # annotations
06. .annotation [ 注解属性] < 注解类名>
07.     [ 注解字段 = 值]
08. .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]

```
01. # instance fields
02. .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”
03.     .annotation runtime Lcom/droider/anno/MyAnnoField;
04.         info = "Hello my friend"
05.     .end annotation
06. .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]

```
01. class Outer {
02.     class Inner{}
03. }
```

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]

```
01. .class public Lcom/droider/crackme0502/MainActivity$SNChecker;
02. .super Ljava/lang/Object;
03. .source "MainActivity.java"
04.
05. # annotations
06. .annotation system Ldalvik/annotation/EnclosingClass;
07.     value = Lcom/droider/crackme0502/MainActivity;
08. .end annotation
09. .annotation system Ldalvik/annotation/InnerClass;
10.     accessFlags = 0x1
11.     name = "SNChecker"
12. .end annotation
13.
```

评论排行

Android面试题	(2)
Android应用实例之----基	(2)
android.os.AsyncTask<S	(1)
Android中使用HttpClient	(1)
加州求职记	(1)
notifyDataSetChanged()	(1)
Android下使用TCP/IP协	(1)
System.gc()	(1)
Android之使用Http协议	(1)
Android关于SD卡中多层	(1)

推荐文章

- * Android安全防护之旅---带你把Apk混淆成中文语言代码
- * TensorFlow文本摘要生成 - 基于注意力的序列到序列模型
- * 创建后台任务的两种代码模式
- * 一个屌丝程序员的人生（五十九）
- * WKWebView与js交互之完美解决方案
- * 年轻人，“砖砖瓦瓦”不应该成为你的梦想！

最新评论

- android自定义ClockView
哦罢了: 学习了,大神看看我的
http://blog.csdn.net
/iamdingruihaha/arti...
- Android动态加载技术 简单易懂
希尔薇: 学习到了 谢谢up主的分享
- java学习--Libsvm java版代码注
ccluqh: 请问,如果要改变svm的参数的默认值,是不是要在这个函数中直接设置? void _lineparse_...
- System.gc()
shy_snow: 理解了JVM的gc,才能注意到这些,楼主赞一个。
- notifyDataSetChanged() 动态更
长安画师: 老兄~程序代码都是些什么乱七八糟的啊~你提交博客之后都没自己审查一下吗? 粘贴错误啊!
- Android动态加载jar、apk的实现
竹林听夜风: 可以从网络上加载jar 文件吗。jar为什么要定义 借口,不定义可以吗
- Android关于SD卡中多层目录的
sincerehui: 试试看
- Android之使用Http协议实现文件
河软_下载号: timelengthText videoText titleText 这三个参数传的...
- 据说年薪30万的Android程序员必
超级小黑猪: bu cuo ...
- Cocos2d-x学习 (一) : HelloWc
棋烟烟: 没评论好可怜,给你占个沙发。

```

14. # instance fields
15. .field private sn:Ljava/lang/String;
16. .field final synthetic this$0:Lcom/droider/crackme0502/MainActivity;
17.
18. # direct methods
19. .method public constructor
20. <init>(Lcom/droider/crackme0502/MainActivity;Ljava/lang/String;)V
21. .....
22. .end method
23.
24. # virtual methods
25. .method public isRegistered()Z
26. .....
27. .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 OnItemSelectedListener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]
01. public void onCreate(Bundle savedInstanceState) {
02.     super.onCreate(savedInstanceState);
03.     setContentView(R.layout.activity_main);
04.
05.     btnAnno = (Button) findViewById(R.id.btn_annotation);
06.     btnCheckSN = (Button) findViewById(R.id.btn_checksns);
07.     edtSN = (EditText) findViewById(R.id.edt_sn);
08.
09.     btnAnno.setOnClickListener(new OnClickListener() {
10.
11.         @Override
12.         public void onClick(View v) {
13.             getAnnotations();
14.         }
15.     });
16.
17.     btnCheckSN.setOnClickListener(new OnClickListener() {
18.
19.         @Override
20.         public void onClick(View v) {
21.             SNChecker checker = new SNChecker(edtSN.getText().toString());
22.             String str = checker.isRegistered() ? "注册码正确" : "注册码错误";
23.             Toast.makeText(MainActivity.this, str, Toast.LENGTH_SHORT).show();
24.         }
25.     });
26.
27. }

```

```

[java]
01. 反编译如下:
02. .method public onCreate(Landroid/os/Bundle;)V
03.     .locals 2
04.     .parameter "savedInstanceState"
05.     .....
06.     .line 32
07.     iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;->btnAnno:
08.     Landroid/widget/Button;
09.     new-instance v1, Lcom/droider/crackme0502/MainActivity$1; #新建一个
10.     MainActivity$1实例
11.     invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$1;
12.     -><init>(Lcom/droider/crackme0502/MainActivity;)V # 初始化MainActivity$1
13.     实例

```

```
14.         invoke-virtual {v0, v1}, Landroid/widget/Button;
15.     ->setOnClickListener(Landroid/view/View$OnClickListener;)V # 设置按钮点击事件
16.     监听器
17.         .line 40
18.         iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;
19.     ->btnCheckSN:Landroid/widget/Button;
20.         new-instance v1, Lcom/droider/crackme0502/MainActivity$2; #新建一个
21.     MainActivity$2实例
22.         invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$2
23.     -><init>(Lcom/droider/crackme0502/MainActivity;)V; # 初始化MainActivity$2实例
24.         invoke-virtual {v0, v1}, Landroid/widget/Button;
25.     ->setOnClickListener(Landroid/view/View$OnClickListener;)V#设置按钮点击事件
26.     监听器
27.         .line 50
28.         return-void
29.     .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]
01. # annotations
02. .annotation system Ldalvik/annotation/AnnotationDefault;
03.     value = .subannotation Lcom/droider/anno/MyAnnoClass;
04.         value = "MyAnnoClass"
05.     .end subannotation
06. .end annotation
```

除了SuppressWarnings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]
01. public final class R {
02.     public static final class attr { //属性
03.     }
04.     public static final class dimen { //尺寸
05.         public static final int padding_large=0x7f040002;
06.         public static final int padding_medium=0x7f040001;
07.         public static final int padding_small=0x7f040000;
08.     }
09.     public static final class drawable { //图片
10.         public static final int ic_action_search=0x7f020000;
11.         public static final int ic_launcher=0x7f020001;
12.     }
13.     public static final class id { //id标识
14.         public static final int btn_annotation=0x7f080000;
15.         public static final int btn_checksno=0x7f080002;
16.         public static final int edt_sno=0x7f080001;
17.         public static final int menu_settings=0x7f080003;
18.     }
}
```

```
19.     public static final class layout {    // 布局
20.         public static final int activity_main=0x7f030000;
21.     }
22.     public static final class menu {      // 菜单
23.         public static final int activity_main=0x7f070000;
24.     }
25.     public static final class string {    // 字符串
26.         public static final int app_name=0x7f050000;
27.         public static final int hello_world=0x7f050001;
28.         public static final int menu_settings=0x7f050002;
29.         public static final int title_activity_main=0x7f050003;
30.     }
31.     public static final class style {     // 样式
32.         public static final int AppTheme=0x7f060000;
33.     }
34. }
```

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]
01.  Iterator< 对象> <对象名> = <方法返回一个对象列表>;
02.  for (< 对象> <对象名> : <对象列表>) {
03.      [处理单个对象的代码体]
04.  }
05.  或者:
06.  Iterator< 对象> <迭代器> = <方法返回一个迭代器>;
07.  while (<迭代器>.hasNext()) {
08.      <对象> <对象名> = <迭代器>.next();
09.      [处理单个对象的代码体]
10.  }
```

```
[java]
01. .method private iterator()V
02.     .locals 7
03.     .prologue
04.     .line 34
05.     const-string v4, "activity"
06.     invoke-virtual {p0, v4}, Lcom/droider/circulate/MainActivity;->
07.     getSystemService
08.         (Ljava/lang/String;)Ljava/lang/Object; # 获取ActivityManager
09.     move-result-object v0
10.     check-cast v0, Landroid/app/ActivityManager;
11.     .line 35
12.     .local v0, activityManager:Landroid/app/ActivityManager;
13.     invoke-virtual {v0}, Landroid/app/ActivityManager;->getRunningAppProcesses()
14.     Ljava/util/List;
15.     move-result-object v2    #正在运行的进程列表
16.     .line 36
17.     .local v2, psInfos:Ljava/util/List;
18.     "Ljava/util/List<Landroid/app/ActivityManager$RunningAppProcessInfo>;"
19.     new-instance v3, Ljava/lang/StringBuilder; # 新建一个StringBuilder 对象
20.     invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V # 调用
21.     StringBuilder 构造函数
22.     .line 37
23.     .local v3, sb:Ljava/lang/StringBuilder;
24.     invoke-interface {v2}, Ljava/util/List;->iterator()Ljava/util/Iterator;
25.     #获取进程列表的迭代器
26.     move-result-object v4
27.     :goto_0 #迭代循环开始
28.     invoke-interface {v4}, Ljava/util/Iterator;->hasNext()Z #开始迭代
29.     move-result v5
```

```

30.     if-nez v5, :cond_0 # 如果迭代器不为空就跳走
31.     .line 40
32.     invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
33.     String;
34.     move-result-object v4 # StringBuilder转为字符串
35.     const/4 v5, 0x0
36.     invoke-static {p0, v4, v5}, Landroid/widget/Toast;->makeText
37.     (Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/
38.     widget/Toast;
39.     move-result-object v4
40.     invoke-virtual {v4}, Landroid/widget/Toast;->show()V # 弹出StringBuilder
41.     的内容
42.     .line 41
43.     return-void # 方法返回
44.     .line 37
45.     :cond_0
46.     invoke-interface {v4}, Ljava/util/Iterator;->next()Ljava/lang/Object;
47.     # 循环获取每一项
48.     move-result-object v1
49.     check-cast v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
50.     .line 38
51.     .local v1, info:Landroid/app/ActivityManager$RunningAppProcessInfo;
52.     new-instance v5, Ljava/lang/StringBuilder; # 新建一个临时的StringBuilder
53.     iget-object v6, v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
54.     ->processName:Ljava/lang/String; # 获取进程的进程名
55.     invoke-static {v6}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
56.     Ljava/lang/String;
57.     move-result-object v6
58.     invoke-direct {v5, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
59.     String;)V
60.     const/16 v6, 0xa #换行符
61.     invoke-virtual {v5, v6}, Ljava/lang/StringBuilder;->append(C)Ljava/
62.     lang/StringBuilder;
63.     move-result-object v5 # 组合进程名与换行符
64.     invoke-virtual {v5}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
65.     String;
66.     move-result-object v5
67.     invoke-virtual {v3, v5}, Ljava/lang/StringBuilder; # 将组合后的字符串添加到
68.     StringBuilder 末尾
69.     ->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
70.     goto :goto_0 #跳转到循环开始处
71. .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]
01. .method private forCirculate()V
02. .locals 8
03. .prologue
04. .line 47
05. invoke-virtual {p0}, Lcom/droider/circulate/MainActivity;-
06. >getApplicationContext()Landroid/content/Context;
07. move-result-object v6
08. invoke-virtual {v6}, Landroid/content/Context; #获取PackageManager
09. ->getPackageManager()Landroid/content/pm/PackageManager;
10. move-result-object v3
11. .line 49
12. .local v3, pm:Landroid/content/pm/PackageManager;
13. const/16 v6, 0x2000
14. .line 48
15. invoke-virtual {v3, v6}, Landroid/content/pm/PackageManager;
16. ->getInstalledApplications(I)Ljava/util/List; #获取已安装的程序列表
17. move-result-object v0
18. .line 50
19. .local v0, appInfos:Ljava/util/List;,"Ljava/util/List<Landroid/content/pm
20. /ApplicationInfo>;"
21. invoke-interface {v0}, Ljava/util/List;->size()I # 获取列表中ApplicationInfo
22. 对象的个数
23. move-result v5
24. .line 51
25. .local v5, size:I
26. new-instance v4, Ljava/lang/StringBuilder; # 新建一个
27. StringBuilder 对象
28. invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V # 调用
29. StringBuilder 的构造函数

```

```
30. .line 52
31. .local v4, sb:Ljava/lang/StringBuilder;
32. const/4 v1, 0x0
33.
34. .local v1, i:I #初始化v1为0
35. :goto_0 #循环开始
36. if-lt v1, v5, :cond_0 #如果v1小于v5, 则跳转到cond_0 标号处
37. .line 56
38. invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/
39. lang/String;
40. move-result-object v6
41. const/4 v7, 0x0
42. invoke-static {p0, v6, v7}, Landroid/widget/Toast; #构造Toast
43. ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
44. Landroid/widget/Toast;
45. move-result-object v6
46. invoke-virtual {v6}, Landroid/widget/Toast;->show()V #显示已安装的程序列表
47. .line 57
48. return-void # 方法返回
49. .line 53
50. :cond_0
51. invoke-interface {v0, v1}, Ljava/util/List;->get(I)Ljava/lang/Object;
52. # 单个ApplicationInfo
53. move-result-object v2
54. check-cast v2, Landroid/content/pm/ApplicationInfo;
55. .line 54
56. .local v2, info:Landroid/content/pm/ApplicationInfo;
57. new-instance v6, Ljava/lang/StringBuilder; # 新建一个临时StringBuilder对象
58. iget-object v7, v2, Landroid/content/pm/ApplicationInfo;->packageName:
59. Ljava/lang/String;
60. invoke-static {v7}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
61. Ljava/lang/String;
62. move-result-object v7 # 包名
63. invoke-direct {v6, v7}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
64. String;)V
65. const/16 v7, 0xa #换行符
66. invoke-virtual {v6, v7}, Ljava/lang/StringBuilder;->append(C)Ljava/
67. lang/StringBuilder;
68. move-result-object v6 # 组合包名与换行符
69. invoke-virtual {v6}, Ljava/lang/StringBuilder;->toString()Ljava/lang
70. /String; #转换为字符串
71. move-result-object v6
72. invoke-virtual {v4, v6}, Ljava/lang/StringBuilder; -
73. >append(Ljava/lang/String;)Ljava/lang/StringBuilder; # 添加到循环外 的
StringBuilder 中
74. .line 52
75. add-int/lit8 v1, v1, 0x1 #下一个索引
76. goto :goto_0 #跳转到循环起始处
77. .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]
01. .method private packedSwitch(I)Ljava/lang/String;
02. .locals 1
03. .parameter "i"
04. .prologue
05. .line 21
06. const/4 v0, 0x0
07. .line 22
08. .local v0, str:Ljava/lang/String; #v0为字符串, 0表示null
09. packed-switch p1, :pswitch_data_0 #packed-switch分支, pswitch_data_0指
10. 定case区域
11. .line 36
12. const-string v0, "she is a person" #default分支
13. .line 39
14. :goto_0 #所有case的出口
15. return-object v0 #返回字符串v0
16. .line 24
17. :pswitch_0 #case 0
```



```
18.     const-string v0, "she is a baby"
19.     .line 25
20.     goto :goto_0  #跳转到goto_0标号处
21.     .line 27
22.     :pswitch_1    #case 1
23.     const-string v0, "she is a girl"
24.     .line 28
25.     goto :goto_0  #跳转到goto_0标号处
26.     .line 30
27.     :pswitch_2    #case 2
28.     const-string v0, "she is a woman"
29.     .line 31
30.     goto :goto_0  #跳转到goto_0标号处
31.     .line 33
32.     :pswitch_3    #case 3
33.     const-string v0, "she is an obasan"
34.     .line 34
35.     goto :goto_0  #跳转到goto_0标号处
36.     .line 22
37.     nop
38.     :pswitch_data_0
39.     .packed-switch 0x0    #case 区域, 从0开始, 依次递增
40.         :pswitch_0    #case 0
41.         :pswitch_1    #case 1
42.         :pswitch_2    #case 2
43.         :pswitch_3    #case 3
44.     .end packed-switch
45. .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]
01. .method private tryCatch(ILjava/lang/String;)V
02.     .locals 10
03.     .parameter "drumsticks"
04.     .parameter "peple"
05.     .prologue
06.     const/4 v9, 0x0
07.     .line 19
08.     :try_start_0  # 第1个try开始
09.     invoke-static {p2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
10.     #将第2个参数转换为int 型
11.     :try_end_0    # 第1个try结束
12.     .catch Ljava/lang/NumberFormatException; {:try_start_0 .. :try_end_0} :
13.     catch_1 # catch_1
14.     move-result v1  #如果出现异常这里不会执行, 会跳转到catch_1标号处
15.     .line 21
16.     .local v1, i:I    #.local声明的变量作用域在.local声明与.end local 之间
17.     :try_start_1  #第2个try 开始
18.     div-int v2, p1, v1  # 第1个参数除以第2个参数
19.     .line 22
20.     .local v2, m:I    #m为商
21.     mul-int v5, v2, v1  #m * i
22.     sub-int v3, p1, v5  #v3 为余数
23.     .line 23
24.     .local v3, n:I
25.     const-string v5, "\u5171\u6709%\u53ea\u9e21\u817f\u0c%\u4e2a\u4eba\u5e73\u5206\u0c\u6bcf\u4eba\u53ef\u5206\u5f97%\u53ea\u0c\u8fd8\u5269\u4e0b%\u53ea"  # 格式化字符串
26.     const/4 v6, 0x4
27.     new-array v6, v6, [Ljava/lang/Object;
28.     const/4 v7, 0x0
29.     .line 24
30.     invoke-static {p1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
31.     move-result-object v8
32.     aput-object v8, v6, v7
33.     const/4 v7, 0x1
34.     invoke-static {v1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
35.     move-result-object v8
```



```
38.      aput-object v8, v6, v7
39.      const/4 v7, 0x2
40.      invoke-static {v2}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
41.      move-result-object v8
42.      aput-object v8, v6, v7
43.      const/4 v7, 0x3
44.      invoke-static {v3}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
45.      move-result-object v8
46.
47.      aput-object v8, v6, v7
48.      .line 23
49.      invoke-static {v5, v6}, Ljava/lang/String;
50.          ->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;
51.      move-result-object v4
52.      .line 25
53.      .local v4, str:Ljava/lang/String;
54.      const/4 v5, 0x0
55.      invoke-static {p0, v4, v5}, Landroid/widget/Toast;
56.          ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
57.          Landroid/widget/Toast;
58.      move-result-object v5
59.      invoke-virtual {v5}, Landroid/widget/Toast;->show()V # 使用Toast 显示格
60.      式化后的结果
61.      :try_end_1 #第2个try 结束
62.      .catch Ljava/lang/ArithmeticException; {:try_start_1 .. :try_end_1} :
63.      catch_0 # catch_0
64.      .catch Ljava/lang/NumberFormatException; {:try_start_1 .. :try_end_1} :
65.      catch_1 # catch_1
66.      .line 33
67.      .end local v1 #i:I
68.      .end local v2 #m:I
69.      .end local v3 #n:I
70.      .end local v4 #str:Ljava/lang/String;
71.      :goto_0
72.      return-void # 方法返回
73.      .line 26
74.      .restart local v1 #i:I
75.      :catch_0
76.      move-exception v0
77.      .line 27
78.      .local v0, e:Ljava/lang/ArithmeticException;
79.      :try_start_2 #第3个try 开始
80.      const-string v5, "\u4eba\u6570\u4e0d\u80fd\u4e3a0" #“人数不能为0”
81.      const/4 v6, 0x0
82.      invoke-static {p0, v5, v6}, Landroid/widget/Toast;
83.          ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
84.          Landroid/widget/Toast;
85.      move-result-object v5
86.      invoke-virtual {v5}, Landroid/widget/Toast;->show()V # 使用Toast 显示异
87.      :try_end_2 #第3个try 结束
88.      .catch Ljava/lang/NumberFormatException; {:try_start_2 .. :try_end_2} :
89.      catch_1
90.      goto :goto_0 #返回
91.      .line 29
92.      .end local v0 #e:Ljava/lang/ArithmeticException;
93.      .end local v1 #i:I
94.      :catch_1
95.      move-exception v0
96.      .line 30
97.      .local v0, e:Ljava/lang/NumberFormatException;
98.      const-string v5, "\u65e0\u6548\u7684\u6570\u5b57\u7b26\u4e32"
99.      #“无效的数值字符串”
100.     invoke-static {p0, v5, v9}, Landroid/widget/Toast;
101.         ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
102.         Landroid/widget/Toast;
103.     move-result-object v5
104.     invoke-virtual {v5}, Landroid/widget/Toast;->show()V # 使用Toast 显示异
105.     常原因
106.     goto :goto_0 #返回
107. .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。

顶 0 踩 0

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

更多文章



电脑椅



大阪机场



插画学习



extract



虚拟办公室



哈佛大学入学

参考知识库



Android知识库

32825 关注 | 2675 收录



Java SE知识库

25257 关注 | 477 收录



Java EE知识库

17135 关注 | 1273 收录



Java 知识库

24998 关注 | 1456 收录



算法与数据结构知识库

15204 关注 | 2320 收录

猜你在找

Javascript面向对象特效&框架封装实战

征服JavaScript高级程序设计与应用实例视频课程

Android高级程序开发

深入Javascript字符串实战视频课程

Android入门实战教程

AndroidSmali语法

AndroidSmali高亮语法插件

smali语法及参考说明

smali语法

apk反汇编之smali语法

7


议价
专业品质供应锦福电器
防火电缆线槽（图）

8


24.00/米
供应网格桥架、网格走
线架、网状走线架

9


17.50/米
厂家供应玻璃钢水槽
优质玻璃钢水槽厂家

[查看评论](#)

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

[全部主题](#)
[Hadoop](#)
[AWS](#)
[移动游戏](#)
[Java](#)
[Android](#)
[iOS](#)
[Swift](#)
[智能硬件](#)
[Docker](#)
[OpenStack](#)
[VPN](#)
[Spark](#)
[ERP](#)
[IE10](#)
[Eclipse](#)
[CRM](#)
[JavaScript](#)
[数据库](#)
[Ubuntu](#)
[NFC](#)
[WAP](#)
[jQuery](#)
[BI](#)
[HTML5](#)
[Spring](#)
[Apache](#)
[.NET](#)
[API](#)
[HTML](#)
[SDK](#)
[IIS](#)
[Fedora](#)
[XML](#)
[LBS](#)
[Unity](#)
[Splashtop](#)
[UML](#)
[components](#)
[Windows Mobile](#)
[Rails](#)
[QEMU](#)
[KDE](#)
[Cassandra](#)
[CloudStack](#)
[FTC](#)
[coremail](#)
[OPhone](#)
[CouchBase](#)
[云计算](#)
[iOS6](#)
[Rackspace](#)
[Web App](#)
[SpringSide](#)
[Maemo](#)
[Compuware](#)
[大数据](#)
[aptech](#)
[Perl](#)
[Tornado](#)
[Ruby](#)
[Hibernate](#)
[ThinkPHP](#)
[HBase](#)
[Pure](#)
[Solr](#)
[Angular](#)
[Cloud Foundry](#)
[Redis](#)
[Scala](#)
[Django](#)
[Bootstrap](#)

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#)
[杂志客服](#)
[微博客服](#)
webmaster@csdn.net
[400-600-2320](tel:400-600-2320) | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved 