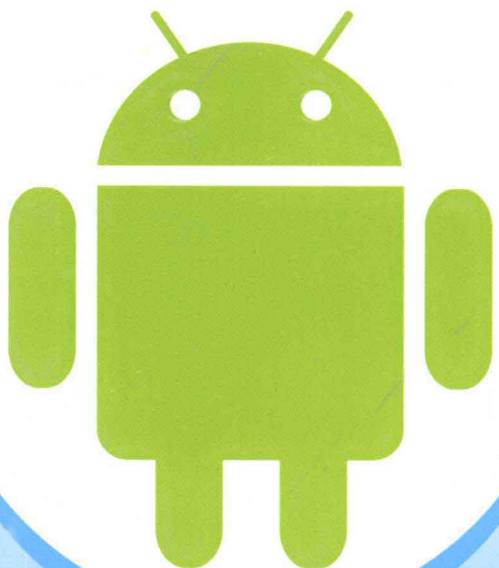
移动互联应用开发系列

Android 项目实战

——手机安全卫士开发案例解析

王家林　王家俊　王家虎◎著



特别分享手机安全卫士软件
Android 源代码供读者下载

通过对开发案例的详细解析
讲解完整 Android 项目的开发过程
将代码中的关键部分逐一解释
有助于读者掌握相关概念和知识

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

移动互联应用开发系列

Android 项目实战

——手机安全卫士开发案例解析

王家林　王家俊　王家虎　著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书通过对一款手机安全卫士开发案例的详细解析，讲解了一个完整的 Android 实际项目的开发过程。该项目涵盖了市场上主流手机卫士的主要功能，同时，该项目也是对 Android 应用程序开发知识的综合应用。通过对案例的解析，使 Android 应用开发人员在实际开发中少走弯路，快速而轻松地积累实战项目经验。

本书适合具备一定的 Android 应用开发知识并需要提高实战开发经验的人员阅读。

图书在版编目（CIP）数据

Android 项目实战：手机安全卫士开发案例解析/王家林，王家俊，王家虎著. —北京：电子工业出版社，2013.5
（移动互联应用开发系列）
ISBN 978-7-121-20084-7

Ⅰ. ①A… Ⅱ. ①王… ②王… ③王… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2013）第 064816 号

责任编辑：窦 昊
印 刷：三河市鑫金马印装有限公司
装 订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1 092 1/16 印张：20.25 字数：518 千字
印 次：2013 年 5 月第 1 次印刷
印 数：4 000 册 定价：49.90 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

为什么要写这样一本书

对于一名希望能够胜任实际开发工作的 Android 应用开发人员来说，最重要的一点是什么呢？毫无疑问，就是积累丰富的项目开发经验，让自己在实际开发工作中游刃有余。

为了让那些希望能够胜任实战工作的 Android 应用开发人员少走弯路，快速而轻松地积累实战项目经验，笔者决定结合多年的项目实战开发经验，编写一本能够真正让读者学以致用的图书。

本书有何特色

为了让读者轻松地上手，本书特别设计了适合初级 Android 应用开发者的学习方式，用准确的定义总结概念，用直观的图示演示过程，用详细的注释解释代码，用简要的语言概括知识点。

①项目模块介绍

简洁、清晰是其显著特点，一般放在每一个模块的开始部分，让读者对每一个模块都有一个清晰、全局的认识。

②代码解析

将代码中的关键代码行逐一解释，有助于读者掌握相关概念和知识。

③运行结果

对每个模块均给出运行结果和对应图示，帮助读者更直观地理解实例代码。

④知识点小结

每完成一个模块，都会对本模块需要掌握的要点进行知识汇总。

本书适合哪些读者阅读

- 具备一定 Android 应用开发基础知识的学习人员；
- 了解 Android 应用开发基础知识，但还需要进一步学习的人员；
- 即将踏入（刚踏入）工作岗位、希望积累项目经验的开发人员；
- 其他编程爱好者。

项目介绍

本书通过对一款手机安全卫士开发案例的详细解析，讲解一个完整的 Android 实际项目的开发过程。该项目涵盖了市场上主流手机卫士的主要功能，同时，该项目也是对 Android 应用程序开发知识的综合应用。项目具体实现了九大功能：手机防盗、通信卫士、软件管理、进程管理、流量统计、手机杀毒、系统优化、高级工具、设置中心。

手机防盗：根据手机 sim 卡的变更来判断手机是否被盗，如果 sim 卡发生变更，程序会根据事先约定好的协议向绑定的安全号码发送一些信息（例如，“sim card changed”等信息），我们可以通过安全号码给手机发送一些指令（例如，远程锁频、销毁数据、播放报警音乐、获取当前手机的经纬度信息），也可以防止应用程序被卸载（例如，将“手机防盗”字样修改为“MP3”）。

通信卫士：来电黑名单管理（电话拦截、短信拦截、电话和短信拦截）。

软件管理：动态计算出当前手机的可用内存、Sdcard 可用内存。同时，以列表的形式显示手机中所有的应用程序（图标、名称、版本号），单击每一个应用程序时，弹出卸载、启动、分享的选项。

进程管理：列出手机中当前正在运行的所有进程，可以将其分为系统进程和用户进程，可以实现对进程的一键清理。

流量管理：显示手机中每个具有 Internet 权限应用程序的上传与下载所产生的流量及流量总和。

手机杀毒：根据程序特征码来识别、查杀病毒。

系统优化：清理应用程序在手机中产生的缓存文件。

高级工具：手机号码归属地查询、常用号码查询、程序锁。

设置中心：是否开启程序的自动更新、是否开启来电归属地的显示、更改归属地的显示风格、更改归属地在屏幕上的显示位置、是否开启黑名单服务、是否开启程序锁服务。

项目的实现流程及说明

项目的基本实现流程是：Splash 界面开发→手机防盗→设置中心→通信卫士→软件管理→进程管理→流量管理→手机杀毒→系统优化。

说明：项目的开发环境是在 Android4.2 环境下进行的，可在 Android2.2 及 Android2.2 以上的版本上运行。在项目实现的过程中，模块与模块之间存在一定的联系，所以在开发的过程中，有时需要进行模块间的切换（例如，在开发手机防盗功能时，需要在设置中心中设置手机防盗是否开启）。

本书的源代码及部分代码文本文件可在电子工业出版社官网“在线资源”中下载，也可以关注微博@电子社通信分社，进入微盘下载相关源码。

笔者

目录 CONTENTS

第 1 章　项目简介与 Splash 界面开发

1.1　创 建 应 用

创建工程文件，应用名称为“手机安全卫士”，工程名为“mobilesafe”，应用包名为“com.guoshisp.mobilesafe”，紧接着将要创建的 Activity 命名为 SplashActivity，该 Activity 用于向用户展现一个 Splash 界面。“Splash”在英文中被译为飞洒、飞溅。

Splash 界面的主要作用：

（1）展现产品的 LOGO，提升产品的知名度。

（2）初始化的操作（初始化数据库、文件的复制、配置的读取）。

（3）根据系统的时间或者日期做出相应的判断来加载不同的 Splash 界面（例如，QQ 的登录界面），提升用户体验。

（4）连接服务器，检查获取更新信息，提示用户升级。在我们的项目中是用于连接服务器，检查版本是否需要更新下载，以及初始化数据库。

新建 Android 项目 mobilesafe，如图 1-1 所示。将 MainActivity 改名为 SplashActivity，如

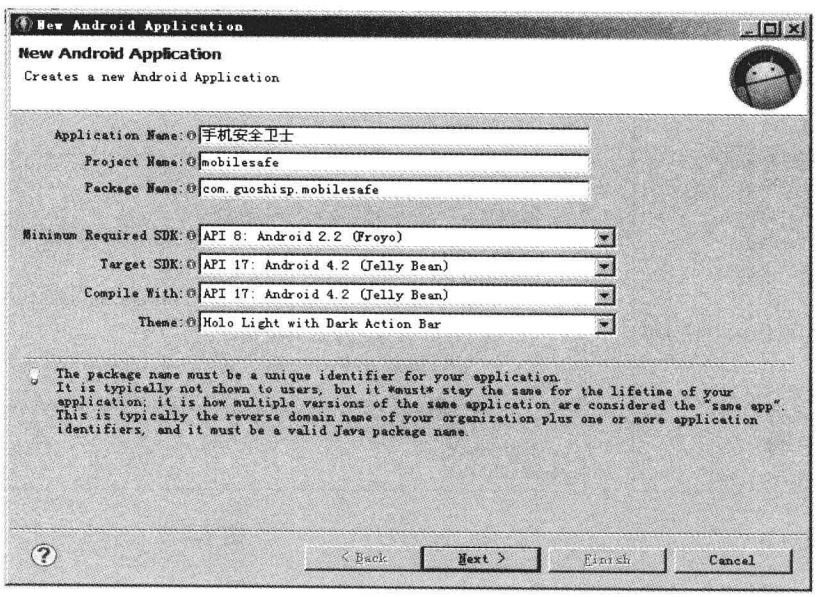


图 1-1

图 1-2 所示。在 res 目录下新建一个 drawable 目录，并将“appicon.png”图片复制到 drawable 文件中（创建该文件的目的在于：原本我们是需要提供三套图片资源来进行屏幕的适配，如果创建了该文件，只需要一套图片资源即可）作为应用的图标，如图 1-3 所示。

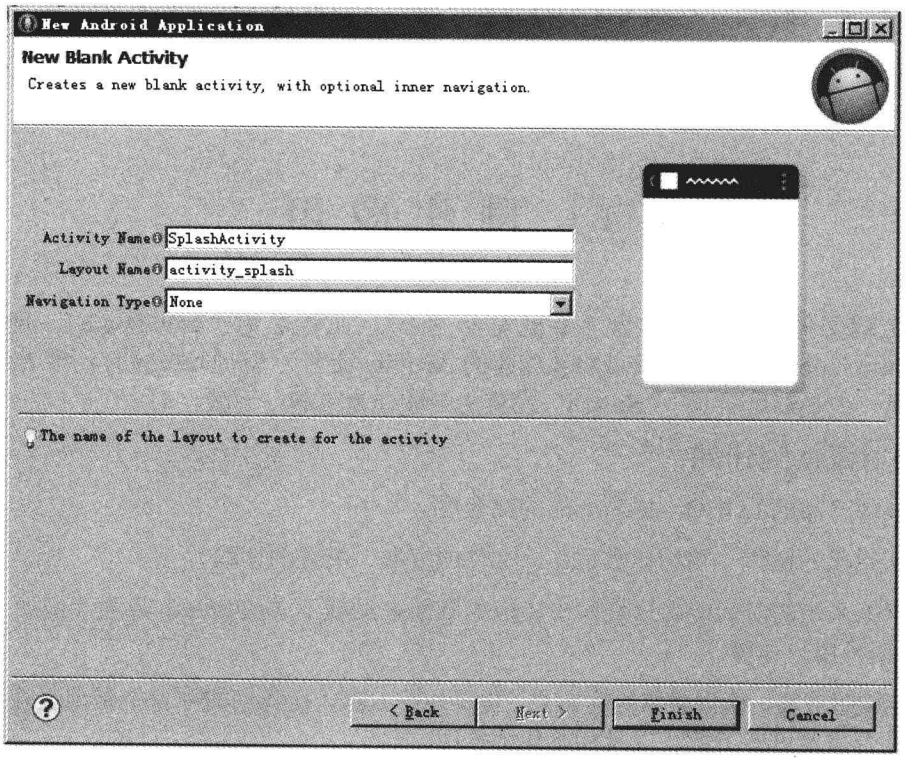


图 1-2

接下来在清单文件中修改应用的图标，如图 1-4 所示。

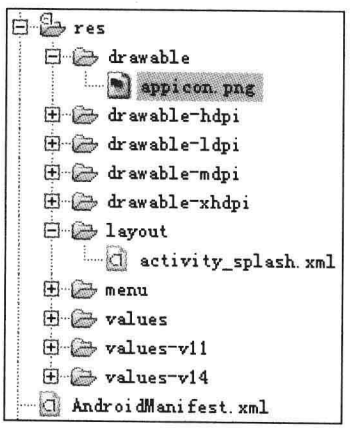


图 1-3

```
<application
    android:allowBackup="true"
    android:icon="@drawable/appicon"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.guoshisp.mobilesafe.SplashActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

图 1-4

1.1.1 Splash 界面的 UI 开发

在 activity_splash.xml 中实现 Splash 界面的 UI 展示。Splash 界面的效果是：将图片 logo2 复制到 drawable 下作为界面的背景图片，在界面的右上角展示当前应用的版本号信息，在界面的中央显示一个 ProgressBar。另外该界面在屏幕上显示时无标题栏且全屏显示（这里是在 SplashActivity 中实现该设置的，当然也可以在清单文件中配置）。效果图如图 1-5 所示。

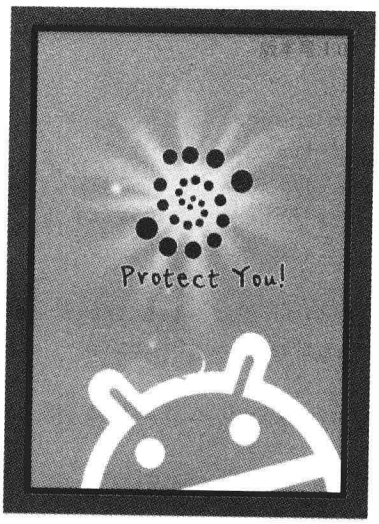


图 1-5

图 1-5 对应的 activity_splash.xml 的 UI 界面代码如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SplashActivity"
    android:id="@+id/rl_splash"
    android:background="@drawable/logo2" >
    <TextView
        android:id="@+id/tv_splash_version"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:text="版本号:"
        android:textColor="#0aff00"
        android:textSize="20sp" />
    <ProgressBar
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="110dip"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

图 1-5 对应的 SplashActivity 业务代码如下：

```
package com.guoshisp.mobilesafe;
import android.app.Activity;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;
import android.widget.RelativeLayout;
import android.widget.TextView;
public class SplashActivity extends Activity {
    private TextView tv_splash_version;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //设置为无标题栏
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        //设置为全屏模式
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.activity_splash);
                tv_splash_version = (TextView) findViewById(R.id.tv_splash_
                                    version);
        tv_splash_version.setText("版本号:" + getVersion());
    }
    /**
     * 获取当前应用程序的版本号
     *
     * @return
     */
    private String getVersion() {
    //得到系统的包管理器，已经得到了 apk 的面向对象的包装
        PackageManager pm = this.getPackageManager();
        try {
    //参数一：当前应用程序的包名；参数二：可选的附加消息，这里用不到，可以定义为 0
            PackageInfo info = pm.getPackageInfo(getPackageName(), 0);
            //返回当前应用程序的版本号
            return info.versionName;
        } catch (Exception e) {//包名未找到异常，理论上，该异常不可能会发生
            e.printStackTrace();
            return "";
        }
    }
}
```

代码解析：

通过调用 getVersion()方法来获取应用程序的版本号信息。版本号存在于我们的 apk 对应的清单文件中（直接解压 apk 后，即可看到对应的清单文件），版本号是 manifest 节点中的 android:versionName="1.0"。当一个应用程序被装到手机后，该 apk 被复制到手机的 data/app 目录下（也就是系统中），如图 1-6 所示。所以想得到版本号，就需要先得到与系统相关的服务（这里我们通过上、下文得到 PackageManager 系统服务），通过服务就可以得到 apk 中的信息。

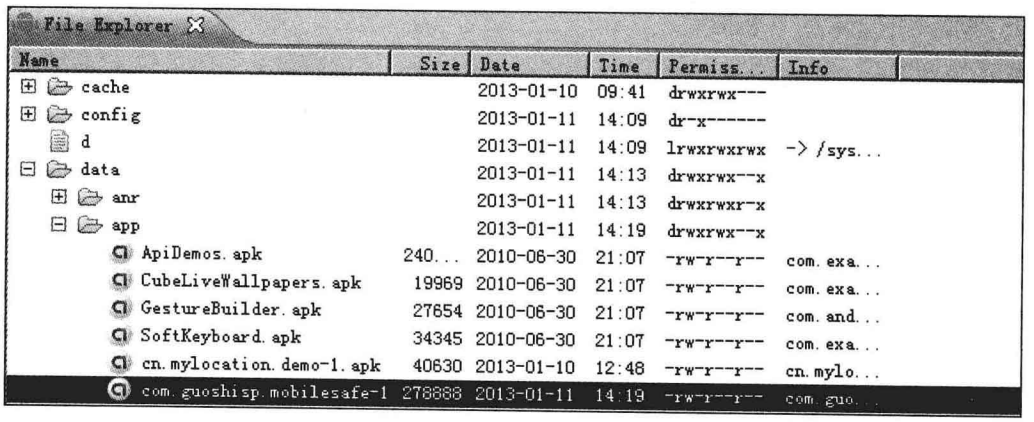


图 1-6

1.1.2 Splash 界面加载时的具体流程

接下来，我们来处理 Splash 界面在加载进来时的具体实现方式：

（1）为 Splash 界面做一个淡进（由暗变明）的动画效果，动画播放时间为 2 秒。

（2）在 Splash 界面加载时，连接服务器检查软件是否需要更新（后面还需要实现对数据库的复制），其流程图如图 1-7 所示。

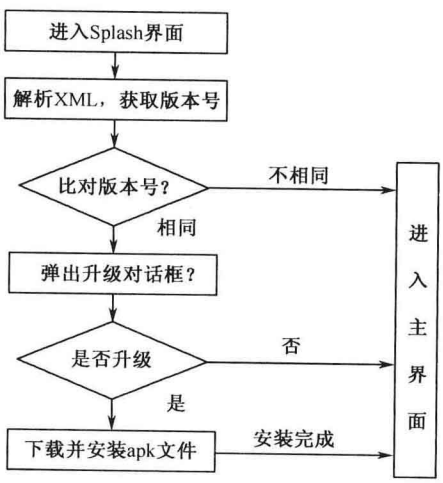


图 1-7

1.1.3 服务器端的搭建

当解析服务端的配置文件时，服务端的配置文件应当包含以下三条信息：

- 版本号，即 versionName。如果版本号为 2.0，请不要将其写为 2；
- 版本升级描述信息，即 description，用于提示用户升级的信息；
- 新版本的 apk 的下载路径。

我们将服务端的配置信息以 XML 文件格式进行存储。这里使用 Tomcat 作为我们的服务器。配置文件存放在 Tomcat 的 ROOT 目录下，文件为 info.xml（在编写该文件时需要注意一点：编写文件时采用的编码方式需要与 encoding 保持一致，否则在使用浏览器解析时会出现字符解析错误，这里采用的是 utf-8 的编码格式），文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<info>
    <version>1.0</version>
    <description>亲，有新版本了，赶紧来下载吧！</description>
    <apkurl>http://192.168.0.4:8080/mobilesafe.apk</apkurl>
</info>
```

同时将最新的 apk 复制到 Tomcat 的 ROOT 目录下。如果服务端的配置信息中的 version 为 2.0，那么所对应的 apk 中的 versionCode="2"，versionName="2.0"。如果 versionName="1.0"，而且本地的 apk 中也是 1.0，即使更新安装后，下次进入时还会提示升级。这是因为本地的 versionName 和服务端的 version 的值不相同。

1.1.4 连接服务器获取更新信息

1. SplashActivity.java 对应的核心业务代码

```
Public class SplashActivity extends Activity {
private UpdateInfo info;
private static final int GET_INFO_SUCCESS = 10;
private static final int SERVER_ERROR = 11;
private static final int SERVER_URL_ERROR = 12;
private static final int PROTOCOL_ERROR = 13;
private static final int IO_ERROR = 14;
private static final int XML_PARSE_ERROR = 15;
protected static final String TAG = "SplashActivity";
private long startTime;
private RelativeLayout rl_splash;
private long endTime;
private Handler handler = new Handler() {
    public void handleMessage(android.os.Message msg) {
        switch (msg.what) {
```

```
            case XML_PARSE_ERROR:
                Toast.makeText(getApplicationContext(), "xml 解析错误",1).show();
                break;
            case IO_ERROR:
                Toast.makeText(getApplicationContext(), "I/O 错误", 1).show();
                break;
            case PROTOCOL_ERROR:
                Toast.makeText(getApplicationContext(), "协议不支持", 1).show();
                break;
            case SERVER_URL_ERROR:
                Toast.makeText(getApplicationContext(), "服务器路径不正确",
                        1).show();
               break;
            case SERVER_ERROR:
                Toast.makeText(getApplicationContext(), "服务器内部异常",
                        1).show();
                break;
            case GET_INFO_SUCCESS:
                String serverversion = info.getVersion();
                String currentversion = getVersion();
                if (currentversion.equals(serverversion)) {
                    Log.i(TAG, "版本号相同进入主界面");
                } else {
                    Log.i(TAG, "版本号不相同,升级对话框");
                    showUpdateDialog();//显示升级对话框
                }
                break;
            }
        };
    };
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //设置为无标题栏
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    //设置为全屏模式
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    setContentView(R.layout.activity_splash);
    rl_splash = (RelativeLayout) findViewById(R.id.rl_splash);
    tv_splash_version = (TextView) findViewById(R.id.tv_splash_version);
    tv_splash_version.setText("版本号:" + getVersion());
    AlphaAnimation aa = new AlphaAnimation(0.3f, 1.0f);
    aa.setDuration(2000);
    rl_splash.startAnimation(aa);
    //连接服务器获取服务器上的配置信息
```

```
        new Thread(new CheckVersionTask()) {
        }.start();
    }
    /**
     * 连网检查应用的版本号与服务端上的版本号是否相同*
     */
    private class CheckVersionTask implements Runnable {
        public void run() {
            startTime = System.currentTimeMillis();
            Message msg = Message.obtain();
            try {
                //获取服务端的配置信息的连接地址
                String serverurl = getResources().getString(R.string. serverurl);
                URL url = new URL(serverurl);
                HttpURLConnection conn = (HttpURLConnection) url
                        .openConnection();
                conn.setRequestMethod("GET");//设置请求方式
                conn.setConnectTimeout(5000);
                int code = conn.getResponseCode();//获取响应码
                if (code == 200) {//响应码为 200 时，表示与服务端连接成功
                    InputStream is = conn.getInputStream();
                    info = UpdateInfoParser.getUpdateInfo(is);
                    endTime = System.currentTimeMillis();
                    long resulttime = endTime - startTime;
                    if (resulttime < 2000) {
                        try {
                            Thread.sleep(2000 - resulttime);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    msg.what = GET_INFO_SUCCESS;
                    handler.sendMessage(msg);
                } else {
                    //服务器状态错误
                    msg.what = SERVER_ERROR;
                    handler.sendMessage(msg);
                    endTime = System.currentTimeMillis();
                    long resulttime = endTime - startTime;
                    if (resulttime < 2000) {
                        try {
                            Thread.sleep(2000 - resulttime);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
```

```
                }
            } catch (MalformedURLException e) {
                e.printStackTrace();
                msg.what = SERVER_URL_ERROR;
                handler.sendMessage(msg);
            } catch (ProtocolException e) {
                msg.what = PROTOCOL_ERROR;
                handler.sendMessage(msg);
                e.printStackTrace();
            } catch (IOException e) {
                msg.what = IO_ERROR;
                handler.sendMessage(msg);
                e.printStackTrace();
            } catch (XmlPullParserException e) {
                msg.what = XML_PARSE_ERROR;
                handler.sendMessage(msg);
                e.printStackTrace();
            }
        }
    }
    /**
     * 显示升级提示的对话框
     */
    protected void showUpdateDialog() {
        //创建对话框的构造器
        AlertDialog.Builder builder = new Builder(this);
    //设置对话框提示标题左边的提示图标
    builder.setIcon(getResources().getDrawable(R.drawable.notification));
        //设置对话框的标题
        builder.setTitle("升级提示");
        //设置对话框的提示内容
        builder.setMessage(info.getDescription());
        //设置升级按钮
        builder.setPositiveButton("升级", new OnClickListener() {
        //设置取消按钮
        public void onClick(DialogInterface dialog, int which) {
        });
        builder.setNegativeButton("取消", new OnClickListener() {
        });
        //创建并显示对话框
        builder.create().show();
    }
```

代码解析：

（1）rl_splash = (RelativeLayout) findViewById(R.id.rl_splash)找到 Splash 界面的根节点，为其播放动画做准备。

（2）为 Splash 界面播放一个动画。

其中，AlphaAnimation aa = new AlphaAnimation(0.3f, 1.0f)设置一个透明度由 0.3f～1.0f 的淡入的动画效果。0.0f 表示完全透明，1.0f 表示正常显示效果。

aa.setDuration(2000)为设置动画的执行时间（0.3f～1.0f），单位为毫秒。

rl_splash.startAnimation(aa)为启动动画。

（3）new Thread(new CheckVersionTask()) {

}.start()

由于连网的过程一般是一个耗时的操作，为了避免出现 ANR 异常，我们在主线程中开启一个子线程用于连网核对版本号信息。此时我们还应当在清单文件中配置网络权限信息：<uses-permission android:name="android.permission.INTERNET"/>。

（4）startTime = System.currentTimeMillis()：用于记录子线程开始执行的时间。

（5）Message msg = Message.obtain()：得到一个向主线程发送消息的消息对象。

（6）String serverurl =getResources().getString(R.string.serverurl)：得到访问服务端配置信息（即服务端的 info.xml 文件）的 URL 地址。

该 URL 地址存放在 values 文件夹下的 config.xml 文件中（将请求服务端的 URL 存放在这里的目的在于：当这个 URL 需要变更时，不需要在代码中进行修改，只需要修改这个配置文件即可实现，降低了开发的成本），代配置信息如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="serverurl">http://192.168.0.4:8080/info.xml</string>
</resources>
```

（7）取得与服务器的连接。

URL url = new URL(serverurl)为使用 serverurl 建立 URL 类对象 HttpURLConnection conn = (HttpURLConnection) url.openConnection()通过 URL 对象打开路径的链接，实际上这个类内部是使用 HttpURLConnection 来实现的，得到 urlConnection 对象 conn.setConnectTimeout(5000)，在 conn 对象中调用 setConnectTimeout()方法，设置链接的超时时间。之所以要设置这个超时时间，是因为：如果请求时间比较长，比如要等 30 秒，那么这个程序就处在一个阻塞过程中，而 Android 组件也有一个阻塞时间，笔者没有精确计算过，但是建议最好不要超过 6 秒，超过 6 秒时，Android 系统就会自动判断，只要超过了它的阻塞时间，不管你的程序是否有错，都会被系统回收。有些请求工作时间比较长的，千万不能在主线程中处理，主线程一旦被阻塞了，一定会被回收。碰到这种情况我们最好抛开主线程去做，而不要在主线程中处理。如果不是开发 Android 应用，而是 J2SE 应用，不设也无所谓，因为它没有超时这个说法。

（8）得到数据。

InputStream is = conn.getInputStream()，用 conn 对象调用 getInputStream()方法得到服务端