Android学习系列(31)--App自动化之使用Ant编译项目多渠道打包

随着工程越来越复杂，项目越来越多，以及平台的迁移(我最近就迁了2回),还有各大市场的发布，自动化编译android项目的需求越来越强烈，后面如果考虑做持续集成的话，会更加强烈。

经过不断的尝试，在ubuntu环境下，以花界为例，我将一步一步演示如何使用命令行，使用ant编译android项目，打包多渠道APK。

要点：

(1). 编译android的命令使用

(2). ant基本应用

(3). 多项目如何编译(包含android library)

(4). 如何多渠道打包

ps：我将以最原始的方式来实现，而不是使用android自带的ant编译方式，并尽量详细解释，这样有益于我们彻底搞懂android打包的基本原理。

1. Android编译打包的整体过程

使用ant，ant的参考文档： http://ant.apache.org/manual/index.html

首先，假设现在已经有这样的一个项目(多工程的，简单的单工程就更简单了)：

```
world
├── baseworld                    //android library，基础类库，共享于其他主应用
├── floworld                     //android project，花界应用
├── healthworld                  //android project，健康视线应用
├── speciality                   //android project，其它应用
├── starworld                    //android project，其它应用
├── build.xml                    //ant编译脚本,可用于整个项目的编译，也可只编译某个工程
```

```
├── code_checks.xml
├── kaiyuanxiangmu_world.keystore          //密钥
└── README.md
```

一个大的项目world，下面有1个基础Android Library和4个Android Project。我们要做的就是编译这4个人project成对应的一系列各市场APK。

那么我们在来看看baseworld和floworld的工程结构:

Android Library,baseworld:

```
baseworld
├── assets                                 //assets目录，其中文件可能会被主应用覆盖
├── libs                                   //存放第三方jar库
├── res                                    //类库资源，其中文件可能会被主应用覆盖
├── src                                    //源码，可直接供主应用使用
├── AndroidManifest.xml
├── lint.xml
├── proguard.cfg
├── project.properties
└── README.md
```

和Android Project,floworld:

```
floworld/
├── assets                                 //assets目录，主应用优先级高
├── build
├── data
├── libs                                   //存放第三方jar库
├── res                                    //主应用资源，主应用优先级高
├── src                                    //源码，可直接供主应用使用
├── AndroidManifest.xml
├── build.xml                              //ant编译脚本,可用于整个项目的编译，也可只编译某个工程
├── default.properties
├── lint.xml
├── proguard.cfg
├── project.properties
└── README.md
```


常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

Android (73)
Android学习系列 (42)
FastObject (20) ExtJs2.0 (16)
Android设计模式系列 (13)
Android拓展系列 (11)
设计模式 (10) Ext (4)
Extjs (4) 个人框架 (3) 更多

随笔分类(70)

Android框架源码剖析系列
Android设计模式系列(13)
Android拓展系列(12)
Android陷阱系列
Android学习系列(39)
Android应用(6)

随笔档案(88)

2016年3月 (1)
2015年11月 (1)
2015年9月 (3)
2015年3月 (1)
2015年1月 (1)
2014年12月 (1)

结构已经出来了，那么android打包主要是在做什么？

说白了，先编译java成class，再把class和jar转化成dex，接着打包aaset和res等资源文件为res.zip(以res.zip示例),再把dex和res.zip合并为一个未签名apk，再对它签名，最终是一个带签名的apk文件。

当然这么说忽略了很多细节。

下面我把这些步骤用一句话分别列举如下，脑子里先有一个整体的流程，后续再结合ant详细展开：

(1). 生成用于主应用的R.java;

(2). 生成用于库应用的R.java(如果有库应用);

(3). 编译所有java文件为class文件;

(4). 打包class文件和jar包为classes.dex;

(5). 打包assets和res资源为资源压缩包(如res.zip，名字可以自己定义);

(6). 组合classes.dex和res.zip生成未签名的APK;

(7). 生成有签名的APK;

针对多项目同步发布和多渠道打包问题，我们需要额外增加三个处理:

(1). 各个工程下建立一个build.xml，然后在整个项目的根目录下建立一个build.xml，用于统一编译各个工程的;

(2). 各个工程的build.xml，通过传入市场ID和应用Version参数生成对应的版本

(3). 针对(1),(2)问题，建立一个批处理支持一键生成所有版本

大概流程即是如此。

2. 建立各个工程的ant脚本文件build.xml(位置:floworld/build.xml)

因为需要创建一些基本的文件目录和清理上次生成的文件，所以我们简单的定义一下几个目标吧：init,main,clean。

代码模板如下:

```
<project default="main" basedir=".">
     <!-- 初始化：创建目录，清理目录等 -->
     <target name="init">
          <echo>start initing ... </echo>
          <!-- ... ... -->
          <echo>finish initing. </echo>
     </target>
     <!-- 打包过程,默认值 -->
     <target name="main" depends="init">
     </target>
     <!-- 清理不需要的生成文件等-->
     <target name="clean">
```

2014年9月 (2)
2014年7月 (1)
2014年5月 (2)
2014年4月 (4)
2013年4月 (2)
2012年12月 (5)
2012年11月 (1)
2012年10月 (1)
2012年9月 (1)
2012年8月 (1)
2012年7月 (2)
2012年6月 (1)
2012年5月 (2)
2012年4月 (2)
2012年3月 (1)
2012年2月 (2)
2012年1月 (1)
2011年12月 (1)
2011年9月 (4)
2011年8月 (9)
2011年7月 (4)
2011年6月 (6)
2011年5月 (4)
2011年4月 (5)
2008年8月 (10)
2008年7月 (2)
2008年6月 (4)

积分与排名

积分 - 389832
排名 - 351

最新评论

1. Re:Android学习系列(33)--…

```
    </target>
</project>
```

3. 初始化

在正式打包之前，有必要说明一下可能需要用到的初始化变量和操作。

前面已经讲述了打包的大概流程，现在，第一, 打包需要你使用哪个版本android.jar; 第二, 生成的R文件放到gen目录下; 第三, 生成的classes文件放到bin目录下; 第四, 生成的打包文件放到out目录下; 第五, 生成的各市场版本放到build目录下。目录完全可以自定义。

所以，如下的初始化必须先要做好，不然后面会提示找不到目录:

```
<project default="main" basedir=".">
    <!-- 这个是android.jar路径，具体情况具体配置 -->
    <property name="android-jar" value="/usr/lib/android-sdk/platforms/android-10/android.
    <!-- 用于生成多渠道版本的APK文件名，提供了默认值，后面会讲到 -->
    <property name="apk-name" value="product" />
    <property name="apk-version" value="latest" />
    <property name="apk-market" value="dev" />
    <target name="init">
        <echo>start initing ... </echo>
        <mkdir dir="out" />
        <delete>
            <fileset dir="out"></fileset>
        </delete>
        <mkdir dir="gen" />
        <delete>
            <fileset dir="gen"></fileset>
        </delete>
        <mkdir dir="bin/classes" />
        <delete>
            <fileset dir="bin/classes"></fileset>
        </delete>
        <!-- ${apk-version}表示版本，后面会详细讲到 -->
        <mkdir dir="build/${apk-version}" />
        <echo>finish initing. </echo>
    </target>
```

```
    ... ...
</project>
```

4. 生成R.java

Android Library和Android Project应用的R.java是来自不同的package的。比如：

(1). baseworld中导入的包是import com.tianxia.lib.baseworld.R;

(2). floworld中导入的包是import com.tianxia.lib.baseworld.R;

但是他们最终是调用统一的资源，所以这两个R.java文件必须一致。

下面是主应用的R.java的生成脚本：

```
<echo>generating R.java for project to dir gen (using aapt) ... </echo>
<exec executable="aapt">
    <arg value="package" /> <!-- package表示打包-->
    <arg value="-m" /> <!--m,J,gen表示创建包名的目录和R.java到gen目录下 -->
    <arg value="-J" />
    <arg value="gen" />
    <arg value="-M" /> <!-- M指定AndroidManifest.xml文件-->
    <arg value="AndroidManifest.xml" />
    <arg value="-S" /> <!-- S指定res目录，生成对应的ID，可多个-->
    <arg value="res" />
    <arg value="-S" />
    <arg value="../baseworld/res" /><!-- 注意点:同时需要调用Library的res-->
    <arg value="-I" /> <!-- I指定android包的位置-->
    <arg value="${android-jar}" />
    <arg value="--auto-add-overlay" /> <!-- 这个重要，覆盖资源，不然报错-->
</exec>
```

注意res和../baseworld/res两个顺序不能搞反，写在前面具有高优先级，我们当然优先使用主应用的资源了，这样就能正确覆盖库应用的资源，实现重写。

库应用的R.java的生成脚本差不多，区别是指定库应用的AndroidManifest.xml，以用于生成的是不同的包和目录。

另外，aapt的使用中特别说明了，为了库应用的资源更好的可重用，库应用生成的R.java字段不需要修饰为final,加上参数--non-constant-id即可。

```
<echo>generating R.java for library to dir gen (using aapt) ... </echo>
```

1. Android学习系列(2)--App自..
2. ExtJs2.0学习系列(1)--Ext....
3. ExtJs2.0学习系列(4)--Ext.F..
4. ExtJs2.0学习系列(2)--Ext....
5. ExtJs2.0学习系列(8)--Ext.F..

推荐排行榜

1. Android学习系列(1)--为Ap...
2. Android学习系列(2)--App自..
3. Android学习系列(27)--App...
4. Android学习系列(7)--App轮..
5. Android学习系列(6)--App模..

```
<exec executable="aapt">
    <arg value="package" />
    <arg value="-m" />
    <arg value="--non-constant-id" /> <!-- 加了这个参数-->
    <arg value="--auto-add-overlay" />
    <arg value="-J" />
    <arg value="gen" />
    <arg value="-M" />
    <arg value="../baseworld/AndroidManifest.xml" /> <!-- 库应用的manifest-->
    <arg value="-S" />
    <arg value="res" />
    <arg value="-S" />
    <arg value="../baseworld/res" />
    <arg value="-I" />
    <arg value="${android-jar}" />
</exec>
```

这样的话就可以生成2个正确的R.java文件了(如果你引用了两个库，则需要生成3个R.java，以此类推)。
结果如下:

```
gen
└── com
    └── tianxia
        ├── app
        │   └── floworld
        │       └── R.java
        └── lib
            └── baseworld
                └── R.java
```

5. 编译java文件为class文件

使用javac命令把src目录,baseworld/src目录,gen/*/R.java这些java编译成class文件：
命令原型是：

```
//示例
```

```
javac -bootclasspath <android.jar> -s <src> -s <src> -s <gen> -d bin/classes *.jar
```

转化成ant脚本为:

```
<!-- 第三方jar包需要引用，用于辅助编译 -->
<path id="project.libs">
    <fileset dir="libs">
        <include name="*.jar" />
    </fileset>
</path>
<echo>compiling java files to class files (include R.java, library and the third-party jar
<!-- 生成的class文件全部保存到bin/classes目录下 -->
<javac destdir="bin/classes" bootclasspath="${android-jar}">
    <src path="../baseworld/src" />
    <src path="src" />
    <src path="gen" />
    <classpath refid="project.libs" />
</javac>
```

6. 打包class文件为classes.dex

这步简单，用dx命令把上步生成的classes和第三方jar包打包成一个classes.dex。
命令原型是:

```
//示例
//后面可以接任意个第三方jar路径
dx --dex --output=out/classes.dex bin/classes libs/1.jar libs/2.jar
```

转化成ant脚本为:

```
<echo>packaging class files (include the third-party jars) to calsses.dex ... </echo>
<exec executable="dx">
    <arg value="--dex" />
    <arg value="--output=out/classes.dex" /><!-- 输出 -->
    <arg value="bin/classes" /> <!-- classes文件位置 -->
    <arg value="libs" /> <!-- 把libs下所有jar打包 -->
```

```
</exec>
```

7. 打包res，assets为资源压缩包(暂且命名为res.zip)

还是使用aapt命令，如生成R.java最大的不同是参数-F,意思是生成res.zip文件。

命令原型和ant脚本差不多:

```
<echo>packaging resource (include res, assets, AndroidManifest.xml, etc.) to res.zip ... <
<exec executable="aapt">
    <arg value="package" />
    <arg value="-f" /> <!-- 资源覆盖重写 -->
    <arg value="-M" />
    <arg value="AndroidManifest.xml" />
    <arg value="-S" />
    <arg value="res" />
    <arg value="-S" />
    <arg value="../baseworld/res" />
    <arg value="-A" /> <!-- 与R.java不同，需要asset目录也打包 -->
    <arg value="assets" />
    <arg value="-I" />
    <arg value="${android-jar}" />
    <arg value="-F" /> <!-- 输出资源压缩包 -->
    <arg value="out/res.zip" />
    <arg value="--auto-add-overlay" />
</exec>
```

8. 使用apkbuilder命令组合classes.dex，res.zip和AndroidManifest.xml为未签名的apk

apkbuilder命令能把class类，资源等文件打包成一个未签名的apk，原型命令和ant脚本类似:

```
<echo>building unsigned.apk ... </echo>
<exec executable="apkbuilder">
    <arg value="out/unsigned.apk" /> <!-- 输出 -->
    <arg value="-u" /> <!-- u指创建未签名的包-->
    <arg value="-z" /> <!-- 资源压缩包 -->
    <arg value="out/res.zip" />
```

```
    <arg value="-f" /> <!-- dex文件 -->
    <arg value="out/classes.dex" />
</exec>
```

这个命令比较简单。

9. 签名未签名的apk

使用jarsigner命令对上步中产生的apk签名。这是个传统的java命令，非android专用。
原型命令和ant脚本差不多:

```
<!-- 生成apk文件到build目录下 -->
<!-- 其中${apk-version/name/market}用户多渠道打包，后面会讲到 -->
<echo>signing the unsigned apk to final product apk ... </echo>
<exec executable="jarsigner">
    <arg value="-keystore" />
    <arg value="../xxx.keystore" />
    <arg value="-storepass" />
    <arg value="xxx" />  <-- 验证密钥完整性的口令，创建时建立的 -->
    <arg value="-keypass" />
    <arg value="xxx" /> <-- 专用密钥的口令，就是key密码 -->
    <arg value="-signedjar" />
    <arg value="build/${apk-version}/${apk-name}_${apk-version}_${apk-market}.apk" /> <!--
    <arg value="out/unsigned.apk" /> <!-- 未签名的apk -->
    <arg value="xxx" /> <!-- 别名,创建时建立的 -->
</exec>
```

至此，完整具有打包功能了，最后的build.xml为：

```
<project default="main" basedir=".">

    <property name="apk-name" value="product" />
    <property name="apk-version" value="latest" />
    <property name="apk-market" value="dev" />

    <property name="android-jar" value="/usr/lib/android-sdk/platforms/android-10/androic
```

```
8
9       <target name="init">
10          <echo>start initing ... </echo>
11
12          <mkdir dir="out" />
13          <delete>
14              <fileset dir="out"></fileset>
15          </delete>
16
17          <mkdir dir="gen" />
18          <delete>
19              <fileset dir="gen"></fileset>
20          </delete>
21
22          <mkdir dir="bin/classes" />
23          <delete>
24              <fileset dir="bin/classes"></fileset>
25          </delete>
26
27          <mkdir dir="build/${apk-version}" />
28
29          <echo>finish initing. </echo>
30      </target>
31
32      <target name="main" depends="init">
33          <echo>generating R.java for project to dir gen (using aapt) ... </echo>
34          <exec executable="aapt">
35              <arg value="package" />
36              <arg value="-m" />
37              <arg value="-J" />
38              <arg value="gen" />
39              <arg value="-M" />
40              <arg value="AndroidManifest.xml" />
41              <arg value="-S" />
```

```
            <arg value="res" />
            <arg value="-S" />
            <arg value="../baseworld/res" />
            <arg value="-I" />
            <arg value="${android-jar}" />
            <arg value="--auto-add-overlay" />
        </exec>

        <echo>generating R.java for library to dir gen (using aapt) ... </echo>
        <exec executable="aapt">
            <arg value="package" />
            <arg value="-m" />
            <arg value="--non-constant-id" />
            <arg value="--auto-add-overlay" />
            <arg value="-J" />
            <arg value="gen" />
            <arg value="-M" />
            <arg value="../baseworld/AndroidManifest.xml" />
            <arg value="-S" />
            <arg value="res" />
            <arg value="-S" />
            <arg value="../baseworld/res" />
            <arg value="-I" />
            <arg value="${android-jar}" />
        </exec>

        <path id="project.libs">
            <fileset dir="libs">
                <include name="*.jar" />
            </fileset>
        </path>
        <echo>compiling java files to class files (include R.java, library and the third-
        <javac destdir="bin/classes" bootclasspath="${android-jar}">
            <src path="../baseworld/src" />
```

```
            <src path="src" />
            <src path="gen" />
            <classpath refid="project.libs" />
        </javac>

        <echo>packaging class files (include the third-party jars) to calsses.dex ... </e
        <exec executable="dx">
            <arg value="--dex" />
            <arg value="--output=out/classes.dex" />
            <arg value="bin/classes" />
            <arg value="libs" />
        </exec>

        <echo>packaging resource (include res, assets, AndroidManifest.xml, etc.) to res.
        <exec executable="aapt">
            <arg value="package" />
            <arg value="-f" />
            <arg value="-M" />
            <arg value="AndroidManifest.xml" />
            <arg value="-S" />
            <arg value="res" />
            <arg value="-S" />
            <arg value="../baseworld/res" />
            <arg value="-A" />
            <arg value="assets" />
            <arg value="-I" />
            <arg value="${android-jar}" />
            <arg value="-F" />
            <arg value="out/res.zip" />
            <arg value="--auto-add-overlay" />
        </exec>

        <echo>building unsigned.apk ... </echo>
        <exec executable="apkbuilder">
```

```
            <arg value="out/unsigned.apk" />
            <arg value="-u" />
            <arg value="-z" />
            <arg value="out/res.zip" />
            <arg value="-f" />
            <arg value="out/classes.dex" />
        </exec>

        <echo>signing the unsigned apk to final product apk ... </echo>
        <exec executable="jarsigner">
            <arg value="-keystore" />
            <arg value="xxx.keystore" />
            <arg value="-storepass" />
            <arg value="xxxx" />
            <arg value="-keypass" />
            <arg value="xxx" />
            <arg value="-signedjar" />
            <arg value="build/${apk-version}/${apk-name}_${apk-version}_${apk-market}.apk
            <arg value="out/unsigned.apk" />
            <arg value="xxx" />
        </exec>

        <echo>done.</echo>
    </target>
</project>
```

在工程目录下运行ant:

```
$ant
Buildfile: build.xml

init:
     [echo] start initing ...
    [mkdir] Created dir: /home/openproject/world/floworld/build/latest
```

```
     [echo] finish initing.

main:
     [echo] generating R.java for project to dir gen (using aapt) ...
     [echo] generating R.java for library to dir gen (using aapt) ...
     [echo] compiling java files to class files (include R.java, library and the third-par
    [javac] Compiling 75 source files to /home/openproject/world/floworld/bin/classes
    [javac] 注意：某些输入文件使用或覆盖了已过时的 API。
    [javac] 注意：要了解详细信息，请使用 -Xlint:deprecation 重新编译。
     [echo] packaging class files (include the third-party jars) to calsses.dex ...
     [echo] packaging resource (include res, assets, AndroidManifest.xml, etc.) to res.zip
     [echo] building unsigned.apk ...
     [exec]
     [exec] THIS TOOL IS DEPRECATED. See --help for more information.
     [exec]
     [echo] signing the unsigned apk to final product apk ...
     [echo] done.

BUILD SUCCESSFUL
Total time: 28 seconds
```

成功的在build/latest目录下生成一个product_latest_dev.apk，这就是默认的生成的最终的APK，可以导入到手机上运行。

10. 多渠道打包

目前主流的多渠道打包方法是在AndroidManifest.xml中的Application下添加一个渠道元数据节点。

比如，我使用的是友盟统计，它配置AndroidManifest.XML添加下面代码：

```
<application ……>
    <meta-data android:value="Channel ID" android:name="UMENG_CHANNEL"/>
    <activity ……/>
</application>
```

通过修改不同的Channel ID值，标识不同的渠道，有米广告提供了一个不错的渠道列表: http://wiki.youmi.net/PromotionChannelIDs .

实现多渠道自动打包，就是实现自动化过程中替换Channel ID，然后编译打包。

这个替换需要用到正则表达式实现。

ant中提供的replace方法，功能太简单了，replaceregrex又需要添加另外的jar包，而且我们后面我们实现ant传参需要写另外的linux shell脚本，所以我干脆使用我熟悉的sed-i命令来实现替换。

替换命令:

```
#-i 表示直接修改文件
#$market是Channel ID, 后面会讲到，是来自循环一个数组
#\1,\3分别表示前面的第1，3个括号的内容，这样写很简洁
sed -i "s/\(android:value=\)\"\(.*\)\"\( android:name=\"UMENG_CHANNEL\"\)/\1\"$market\"\3/g
```

渠道修改的问题解决了。

还记得前面定义的${apk-version},${apk-name},${apk-market}吗？

ant提供了额外的参数形式可以修改build.xml中定义的属性的值：ant -Dapk-version=1.0，则会修改${apk-version}值为1.0，而不是latest了，其他属性类似。

所以，在工程下面这条命令会生成:

```
#结合前面讲打build.xml
#会在build/1.0/目录下生成floworld_1.0_appchina.apk
ant -Dapk-name=floworld -Dapk-version=1.0 -Dapk-market=appchina
```

命令问题通过ant的参数传值也解决了。

现在需要的是批量生产N个市场的版本，既替换AndroidManifest.xml，又生成对应的apk文件，我结合上面说的亮点，写了一个shell脚本(位置:world/floworld/build.sh):

```
#定义市场列表，以空格分割
markets="dev appchina gfan"
#循环市场列表，分别传值给各个脚本
for market in $markets
do
    echo packaging floworld_1.0_$market.apk ...
    #替换AndroidManifest.xml中Channel值(针对友盟,其他同理)
    sed -i "s/\(android:value=\)\"\(.*\)\"\( android:name=\"UMENG_CHANNEL\"\)/\1\"$market\
    #编译对应的版本
    ant -Dapk-name=floworld -Dapk-version=1.0 -Dapk-market=$market
```

```
done
```

好的，在工程目录下执行build.sh:

```
# ./build.sh
packaging floworld_1.0_dev.apk ...
Buildfile: build.xml
... ...
packaging floworld_1.0_appchina.apk ...
Buildfile: build.xml
... ...
packaging floworld_1.0_gfan.apk ...
Buildfile: build.xml
... ...
```

在build下生成了对应的apk文件:

```
build
├── 1.0
│   ├── floworld_1.0_appchina.apk
│   ├── floworld_1.0_dev.apk
│   └── floworld_1.0_gfan.apk
└── README.md
```

成功生成!

11. 工程脚本的执行目录问题

上面的脚本执行之后的确很cool，但是有一个问题，我必须在build.sh目录下执行，才能正确编译，这个和build.xml中定义的相对路径有关。

我们必须在任何目录执行工程目录下的build.sh都不能出错，改进build.sh为如下:

```
#!/bin/bash
#添加如下两行简单的代码
#1. 获取build.sh文件所在的目录
#2. 进入该build.sh所在目录，这样执行起来就没有问题了
basedir=$(cd "$(dirname "$0")";pwd)
```

```
cd $basedir

markets="dev appchina gfan"
for market in $markets
do
    echo packaging floworld_1.0_$market.apk ...
    sed -i "s/\(android:value=\)\"\(.*\)\"\( android:name=\"UMENG_CHANNEL\"\)/\1\"$market\
    ant -Dapk-name=floworld -Dapk-version=1.0 -Dapk-market=$market
done
```

现在你在项目根目录下执行也没有问题:./floworld/build.sh，不会出现路径不对，找不到文件的错误了。

12. 建立整个项目的自动化编译脚本(位置:world/build.sh)

单个工程的自动化打包没有问题了，但是一个项目下有N个工程，他们往往需要同步发布(或者daily build也需要同步编译)，所以有必要建立一个项目级别的编译脚本:

build.sh(项目根目录下，位置:/world/build.sh)

最简单的傻瓜式的做法就是，遍历项目下的工程目录，如果包含工程编译的build.sh,则编译该工程.

shell脚本如下:

```
#!/bin/bash
#确保进入项目跟目录
basedir=$(cd "$(dirname "$0")";pwd)
cd $basedir
#遍历项目下各工程目录
for file in ./*
do
if test -d $file
then
    #进入工程目录
    cd $basedir/$file
    #查找该工程目录下是否存在编译脚本build.sh
    if test -f build.sh
    then
        echo found build.sh in project $file.
```

```
        echo start building project $file ...
        ./build.sh
    fi
    #重要，退出工程目录到项目根目录下
    cd $basedir
fi
done
```

执行该脚本:

```
# ./build.sh
found build.sh in project ./floworld.
start building project ./floworld ...
packaging floworld_1.0_dev.apk ...
Buildfile: build.xml
...
...

found build.sh in project ./healthworld.
start building project ./healthworld ...
Buildfile: build.xml
...
```

成功自动寻找，并编译打包。

13. 其他细节

为了尽量详细，我一再解说，但是还有一些细节未包括其中，如编译后清理clean目标，apk对齐优化，java代码混淆等，请参考其他资料，在此省略。

另外，我反编译生成的apk，查看Androidmanifest.xml均正确对应，验证通过。

14. 小结

自动化编译多渠道打包这个功能是Android产品发布的重要环节，能大大节省人力和出错的概率。

最近写博客的步伐慢了下来了，我决定加大力度，多写几篇。

其实早就想做这一块的内容了，一直不得空，或者说心里不得空，前几天刚完全格式化硬盘迁移到ubuntu(抛弃了双系统), 最近又开始着手准备一套《使用vim开发android》视频教程，人又不能太过于沉溺于舒适区域，我决定逼迫一下自己，终于逼出了这篇文章。

真是不容易，今天好热啊，火，你就是火，到处都是火 ... ...
本人项目中的具体应用示例: https://github.com/openproject/world

分类: Android学习系列

标签: Android , Android学习系列

谦虚的天下
关注 - 28
粉丝 - 2159
荣誉： 推荐博客
+加关注

13 0

« 上一篇： Android学习系列(30)--App列表之下拉刷新
» 下一篇： Android拓展系列(7)--vim编辑器的进阶使用

posted @ 2012-07-04 00:16 谦虚的天下 阅读(26919) 评论(37) 编辑 收藏

评论列表

#1楼 2012-07-04 10:43 iStar

非常好，写得很详细，获益匪浅！

另外我想问一下，像这种一个工程引用另一个工程的资源或者代码，会不会造成打出来的apk包变得很大呢？还是只打包用到的资源？

文中提到的《使用vim开发android》，听起来很疯狂啊，期待中！

支持(0) 反对(0)

#2楼 [楼主] 2012-07-04 11:12 谦虚的天下

@ iStar
只打包用到的资源

支持(0) 反对(0)

#3楼 2012-07-04 11:52 elfin1314

太过犀利~~~

支持(0) 反对(0)

#4楼 2012-07-04 14:54 天明打个盹

哈哈，非常犀利。。顶起！

支持(1) 反对(0)

#5楼 [楼主] 2012-07-04 15:36 谦虚的天下

@ 林梦
帅哥来捧场，承让承让~

支持(0) 反对(0)

#6楼 2012-07-05 09:46 红涛

说得比较详细 顶一个
这个自动化打包 如果 需要用到proguard做代码混淆 应该怎么处理

支持(0) 反对(0)

#7楼 [楼主] 2012-07-05 10:33 谦虚的天下

@ 红涛
我现在没有时间做这个，你参考网上的混淆的命令，应该很容易加上

支持(0) 反对(0)

#8楼 2012-07-16 14:20 smoonthsky

这个可以减少多少 人力啊 顶一个！

支持(0) 反对(0)

#9楼 2012-08-13 18:04 hardiman

写的很好,学习

支持(0) 反对(0)

#10楼 2012-09-18 16:47 hy3112

请教一下，如果baseworld里面有自定义属性，并且 floworld使用了baseworld里定义的自定义属性，生成library的R.java时会报
error: No resource identifier found for attribute '...' in package '...'后面这个package指floworld里的package

支持(0) 反对(0)

#11楼 2012-11-10 13:56 天籁の圓

太感谢分享了，精神可嘉，请你吃饭吧。

支持(2) 反对(0)

#12楼 2012-11-13 16:47 天籁の圓

可以加你请求吗，不会问你太多技术问题的 哈哈~~

支持(0) 反对(0)

#13楼 2012-11-14 17:04 天籁の圓

Mac 下通过。。。嘿嘿
感谢你的分享

但是Mac有一个要注意的问题，是Javac的编码配置

还有就是sed -i 的命令 Mac中比较奇葩 一定要这样 sed -i ‘’
加多两个单冒号，再次感谢楼主

支持(0) 反对(0)

#14楼 [楼主] 2012-11-14 22:24 谦虚的天下

@ 天籁の圓
完全可以 673592063

支持(0) 反对(0)

#15楼 2012-11-18 18:50 renfujiang

这个linux 上的，给位 如果用的是 window 不合适

支持(0) 反对(0)

#16楼 [楼主] 2012-11-22 13:46 谦虚的天下

@ renfujiang
文章中已经说了，是在ubuntu环境中

支持(0) 反对(0)

#17楼 2012-12-25 11:31 蝉翼

小弟专门注册了个账号上来咨询一下各位
有没有在windows下编译多项目(包含android library)
的资料，
比如
1.windows上我就无法使用aapt添加library的res
2.而r.java可以使用“--extra-packages”添加成功，但是最终生成的“.class”都比eclipse生成的小一些，
不知道什么问题
期待大家解惑

支持(0) 反对(0)

#18楼 [楼主] 2012-12-25 12:07 谦虚的天下

@ 蝉翼
windows打包我没有去做，但是网上有一些现成的：
http://www.umeng.com/lab
第二个问题，关于自己生成的.class小的问题，可能是你的debug属性没有打开，以后如果有异常，错误提示里面的source行会变成unknown，所以建议打开；而eclipse默认是打开的。贴上示例代码吧：

```
<echo>compiling java files to class files (include R.java, library and the third-party jars) ... </echo>
<javac destdir="bin/classes" bootclasspath="${android-jar}" debug="true">
<src path="../baseworld2/src" />
<src path="../baseworld2.waps/src" />
<src path="src" />
<src path="gen" />
<classpath refid="project.libs" />
</javac>
```

支持(0) 反对(0)

#19楼 2012-12-26 20:45 蝉翼

谢谢楼主解答，
上面链接那个工具下载了，虽然不知道什么原因没有编译成功，但是通过日志看到里面是对依赖的第三方库先分别做了编译，res好像也没有做特殊处理，还是继续google...

支持(0) 反对(0)

#20楼 2013-02-05 15:45 农民伯伯

mac上跑

```
<echo>packaging class files (include the third-party jars) to calsses.dex ... </echo>
<apply executable="dx">
<arg value="--dex" />
<arg value="--output=out/classes.dex" />
<arg value="bin/classes" />
<arg value="libs" />
</apply>
```

这个apply要改成exec才可以跑通过，否则会报no resources specified

支持(0) 反对(0)

#21楼 [楼主] 2013-02-17 10:29 谦虚的天下

@ 农民伯伯
厉害呀，有这个问题，已经改过来了~

支持(0) 反对(0)

#22楼 2013-03-27 10:49 xrh

好文章

支持(0) 反对(0)

#23楼 2013-03-27 17:54 ①块腹肌

good

支持(0) 反对(0)

#24楼 2013-04-12 16:34 捡破烂的

你太伟大了！！！

支持(0) 反对(0)

#25楼 2013-04-27 16:42 codingstandards

博主，你真是太棒了。我把你提供的build.xml做了点小修改，在Windows下编译成功了，有兴趣的可以看看在Windows下用ant编译Android应用生成apk安装包。

支持(0) 反对(0)

#26楼 2013-06-07 20:57 mppds

太强大了，小弟对正则表达式一窍不通，所以有个问题，在我的项目里 umeng的channel是这么写的：
<meta-data
android:name="UMENG_CHANNEL"
android:value="dailyTest" />
那么这样的话，您例子中的 "s/\(android:value=\)\"\(.*\)\"\(
android:name=\"UMENG_CHANNEL\"\)/\1\"$market\"\3/g"
应该改成什么样呢？非常感谢

支持(0) 反对(0)

#27楼 2013-08-06 10:24 jackrex

你太厉害了，向你学习

支持(0) 反对(0)

#28楼 2013-11-02 11:59 loogoo

求救，东西很好用，不过加上混淆以后，就只能生成一次apk包，不能批量，求指教

支持(0) 反对(0)

#29楼 2014-03-24 16:26 mppds

@ loogoo
再写个shell脚本做循环

支持(0) 反对(0)

#30楼 2014-06-09 19:43 demonolver

感谢分享，终于解决了多工程资源引用的问题
但由于dex生成的classes.dex中只包含了.class文件，但使用的第三方jar包中包含了.properties配置文件，dx是否有参数可指定保留这些文件

支持(0) 反对(0)

#31楼 2014-06-09 19:43 demonolver

补充通过eclipse编译打包是包含这些文件的

支持(0) 反对(0)

#32楼 2014-08-19 17:07 晓灯

楼主我的还是不行啊

支持(0) 反对(0)

#33楼 2014-09-16 18:38 又一威猛的小老虎

楼主太牛逼！

决定明天试一下。

支持(0) 反对(0)

#34楼 2014-10-13 11:46 huhuang03

厉害，决定得空得时候把例子下下来系统弄下

支持(0) 反对(0)

#35楼 2014-12-17 11:05 xiao蜗牛

@ smoonthsky
用Eclipse签名打包不是也很快吗？

支持(0) 反对(0)

#36楼 2015-03-04 15:58 1.曲待续

表示很强大，学习了，谢谢

支持(0) 反对(0)

#37楼 2015-05-29 11:09 yummyUmi

请问window下 设置APK签名，渠道版本，要用什么脚本呢

支持(0) 反对(0)

最新知识库文章:

· 如何阅读计算机科学类的书
· Google 及其云智慧
· 做到这一点，你也可以成为优秀的程序员
· 写给立志做码农的大学生
· 架构腐化之谜
» 更多知识库文章...