Android 开发最佳实践

thinkphp

摘要

从Futurice公司Android开发者中学到的经验。 遵循以下准则，避免重复发明轮子。若您对开发iOS或Windows Phone 有兴趣， 请看**iOS Good Practices** 和 **Windows client Good Practices** 这两篇文章。

摘要

- 使用 Gradle 和它推荐的工程结构
- 把密码和敏感数据放在gradle.properties
- 不要自己写 HTTP 客户端,使用Volley或OkHttp库
- 使用Jackson库解析JSON数据
- 避免使用Guava同时使用一些类库来避免*65k method limit*（一个Android程序中最多能执行65536个方法）
- 使用 Fragments来呈现UI视图
- 使用 Activities 只是为了管理 Fragments
- Layout 布局是 XMLs代码，组织好它们
- 在layoutout XMLs布局时，使用styles文件来避免使用重复的属性
- 使用多个style文件来避免单一的一个大style文件
- 保持你的colors.xml 简短DRY(不要重复自己)，只是定义调色板
- 总是使用dimens.xml DRY(不要重复自己)，定义通用常数
- 不要做一个深层次的ViewGroup
- 在使用WebViews时避免在客户端做处理，当心内存泄露
- 使用Robolectric单元测试，Robotium 做UI测试
- 使用Genymotion 作为你的模拟器
- 总是使用ProGuard 和 DexGuard混淆来项目

Android SDK

将你的Android SDK放在你的home目录或其他应用程序无关的位置。 当安装有些包含SDK的IDE的时候，可能会将SDK放在IDE同一目录下，当你需要升级（或重新安装）IDE或更换的IDE时，会非常麻烦。 此外，若果你的IDE是在普通用户，不是在root下运行，还要避免吧SDK放到一下需要sudo权限的系统级别目录下。

构建系统

你的默认编译环境应该是Gradle. Ant 有很多限制，也很冗余。使用Gradle，完成以下工作很方便：

- 构建APP不同版本的变种
- 制作简单类似脚本的任务
- 管理和下载依赖
- 自定义秘钥
- 更多

同时，Android Gradle插件作为新标准的构建系统正在被Google积极的开发。

工程结构

有两种流行的结构：老的Ant & Eclipse ADT 工程结构，和新的Gradle & Android Studio 工程结构， 你应该选择新的工程结构，如果你的工程还在使用老的结构，考虑放弃吧，将工程移植到新的结构。

老的结构:

```
old-structure
├── assets
├── libs
├── res
├── src
│   └── com/futurice/project
├── AndroidManifest.xml
├── build.gradle
├── project.properties
└── proguard-rules.pro
```

新的结构

```
new-structure
├── library-foobar
├── app
│   ├── libs
│   ├── src
│   │   ├── androidTest
│   │   │   └── java
│   │   │       └── com/futurice/project
│   │   └── main
│   │       ├── java
│   │       │   └── com/futurice/project
│   │       ├── res
│   │       └── AndroidManifest.xml
│   ├── build.gradle
│   └── proguard-rules.pro
├── build.gradle
└── settings.gradle
```

主要的区别在于，新的结构明确的分开了'source sets'（`main`，`androidTest`），Gradle的一个理念。 你可以做到，例如，添加源组‘paid’和‘free’在src中，这将成为您的应用程序的付费和免费的两种模式的源代码。

你的项目引用第三方项目库时（例如，library-foobar），拥有一个顶级包名`app`从第三方库项目区分你的应用程序是非常有用的。 然后`settings.gradle`不断引用这些库项目，其中`app/build.gradle`可以引用。

Gradle 配置

常用结构 参考Google's guide on Gradle for Android

小任务 除了(shell, Python, Perl, etc)这些脚本语言，你也可以使用Gradle 制作任务。 更多信息请参考Gradle's documentation。

密码 在做版本release时你app的 `build.gradle` 你需要定义 `signingConfigs` .此时你应该避免以下内容：

不要做这个 . 这会出现在版本控制中。

```
signingConfigs {
    release {
        storeFile file("myapp.keystore")
        storePassword "password123"
        keyAlias "thekey"
        keyPassword "password789"
    }
}
```

而是，建立一个不加入版本控制系统的 `gradle.properties` 文件。

```
KEYSTORE_PASSWORD=password123
KEY_PASSWORD=password789
```

那个文件是gradle自动引入的，你可以在 `buld.gradle` 文件中使用，例如：

```
signingConfigs {
    release {
        try {
            storeFile file("myapp.keystore")
            storePassword KEYSTORE_PASSWORD
            keyAlias "thekey"
            keyPassword KEY_PASSWORD
        }
        catch (ex) {
            throw new InvalidUserDataException("You should define KEYSTORE_PASSWORD and KEY_PASS
WORD in gradle.properties.")
        }
    }
}
```

使用 Maven 依赖方案代替使用导入jar包方案 如果在你的项目中你明确使用率 jar文件，那么它们可能成为永久的版本，如 `2.1.1` .下载jar包更新他们是很繁琐的， 这个问题Maven很好的解决了，这在Android Gradle构建中也是推荐的方法。你可 以指定版本的一个范围，如 `2.1.+` ,然后Maven会自动升级到制定的最新版本，例如：

```
dependencies {
    compile 'com.netflix.rxjava:rxjava-core:0.19.+'
    compile 'com.netflix.rxjava:rxjava-android:0.19.+'
    compile 'com.fasterxml.jackson.core:jackson-databind:2.4.+'
    compile 'com.fasterxml.jackson.core:jackson-core:2.4.+'
    compile 'com.fasterxml.jackson.core:jackson-annotations:2.4.+'
    compile 'com.squareup.okhttp:okhttp:2.0.+'
    compile 'com.squareup.okhttp:okhttp-urlconnection:2.0.+'
}
```

IDE集成开发环境和文本编辑器

无论使用什么编辑器，一定要构建一个良好的工程结构 编辑器每个人都有自己的 选择，让你的编辑器根据工程结构和构建系统运作，那是你自己的责任。

当下首推Android Studio,因为他是由谷歌开发，最接近Gradle，默认使用最新的工程结构，已经到beta阶段（目前已经有release 1.0了），它就是为Android开发定制的。

你也可以使用Eclipse ADT ，但是你需要对它进行配置，因为它使用了旧的工程结构 和Ant作为构建系统。你甚至可以使用纯文版编辑器如Vim，Sublime Text，或者Emacs。如果那样的话，你需要使用Gardle和 `adb` 命令行。如果使用Eclipse集成Gradle 不适合你，你只是使用命令行构建工程，或迁移到Android Studio中来吧。

无论你使用何种开发工具，只要确保Gradle和新的项目结构保持官方的方式构建应用程序，避免你的编辑器配置文件加入到版本控制。例如，避免加入Ant `build.xml` 文件。 特别如果你改变Ant的配置，不要忘记保持 `build.gradle` 是最新和起作用的。同时，善待其他开发者，不要强制改变他们的开发工具和偏好。

类库

Jackson 是一个将java对象转换成JSON与JSON转化java类的类库。 Gson 是解决这个问题的流行方案，然而我们发现Jackson更高效,因为它支持替代的方法处理JSON:流、内存树模型,和传统JSON-POJO数据绑定。不过，请记住，Jsonkson库比起GSON更大，所以根据你的情况选择，你可能选择GSON来避免APP 65k个方法限制。其它选择: Json-smart and Boon JSON

网络请求，缓存，图片 执行请求后端服务器，有几种交互的解决方案，你应该考虑实现你自己的网络客户端。使用Volley 或Retrofit。Volley 同时提供图片缓存类。若果你选择使用Retrofit,那么考虑使用Picasso 来加载图片和缓存，同时使用OkHttp作为高效的网络请求。Retrofit，Picasso和OkHttp都是有同一家公司开发（注： 是由Square 公司开发），所以它们能很好的在一起运行。OkHttp 同样可以和Volley在一起使用 Volley.

RxJava 是函数式反应性的一个类库，换句话说，能处理异步的事件。 这是一个强大的和有前途的模式，同时也可能会造成混淆，因为它是如此的不同。 我们建议在使用这个库架构整个应用程序之前要谨慎考虑。 有一些项目是使用RxJava完成的，如果你需要帮助可以跟这些人取得联系： Timo Tuominen, Olli Salonen, Andre Medeiros, Mark Voit, Antti Lammi, Vera Izrailit, Juha Ristolainen. 我们也写了一些博客： [1], [2], [3], [4].

如若你之前有使用过Rx的经历，开始从API响应应用它。 另外，从简单的UI事件处理开始运用，如单击事件或在搜索栏输入事件。 若对你的Rx技术有信心，同时想要将它应用到你的整体架构中，那么请在复杂的部分写好Javadocs文档。 请记住其他不熟悉RxJava的开发人员，可能会非常难理解整个项目。 尽你的的全力帮助他们理解你的代码和Rx。

Retrolambda 是一个在Android和预JDK8平台上的使用Lambda表达式语法的Java类库。 它有助于保持你代码的紧凑性和可读性，特别当你使用如RxJava函数风格编程时。 使用它时先安装JDK8，在Android Studio工程结构对话框中把它设置成为SDK路径，同时设置 `JAVA8_HOME` 和 `JAVA7_HOME` 环境变量， 然后在工程根目录下配置build.gradle：

```
dependencies {
    classpath 'me.tatarka:gradle-retrolambda:2.4.+'
}
```

同时在每个module 的build.gradle中添加

```
apply plugin: 'retrolambda'

android {
    compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}

retrolambda {
    jdk System.getenv("JAVA8_HOME")
    oldJdk System.getenv("JAVA7_HOME")
    javaVersion JavaVersion.VERSION_1_7
}
```

Android Studio 提供Java8 lambdas表带是代码提示支持。如果你对lambdas不熟悉，只需参照以下开始学习吧：

- 任何只包含一个接口的方法都是"lambda friendly"同时代码可以被折叠成更紧凑的语法
- 如果对参数或类似有疑问，就写一个普通的匿名内部类，然后让Android Status为你生成一个lambda。

当心dex方法数限制，同时避免使用过多的类库 Android apps，当打包成一个dex文件时，有一个65535个应用方法强硬限制[1] [2] [3]。 当你突破65k限制之后你会看到一个致命错误。因此，使用一个正常范围的类库文件，同时使用dex-method-counts 工具来决定哪些类库可以再65k限制之下使用，特别的避免使用Guava类库，因为它包含超过13k个方法。

Activities and Fragments

Fragments应该作为你实现UI界面默认选择。你可以重复使用Fragments用户接口来 组合成你的应用。我们强烈推荐使用Fragments而不是activity来呈现UI界面，理由如下：

- **提供多窗格布局解决方案** Fragments 的引入主要将手机应用延伸到平板电脑，所以在平板电脑上你可能有A、B两个窗格，但是在手机应用上A、B可能分别充满 整个屏幕。如果你的应用在最初就使用了fragments，那么以后将你的应用适配到其他不同尺寸屏幕就会非常简单。

- **屏幕间数据通信** 从一个Activity发送复杂数据(例如Java对象)到另外一个Activity，Android的API并没有提供合适的方法。不过使用Fragment，你可以使用 一个activity实例作为这个activity子fragments的通信通道。即使这样比Activity与Activity间的通信好，你也想考虑使用Event Bus架构，使用如 Otto 或者 greenrobot EventBus作为更简洁的实现。 如果你希望避免添加另外一个类库，RxJava同样可以实现一个Event Bus。

- **Fragments 一般通用的不只有UI** 你可以有一个没有界面的fragment作为Activity提供后台工作。 进一步你可以使用这个特性来创建一个fragment 包含改变其它fragment的逻辑 而不是把这个逻辑放在activity中。

- **甚至ActionBar 都可以使用内部fragment来管理** 你可以选择使用一个没有UI界面的fragment来专门管理ActionBar,或者你可以选择使用在每个Fragment中 添加它自己的action 来作为父Activity的ActionBar.参考.

很不幸，我们不建议广泛的使用嵌套的fragments，因为 有时会引起matryoshka bugs。我们只有当它有意义(例如，在水平滑动的ViewPager在 像屏幕一样fragment中)或者他的确是一个明智的选择的时候才广泛的使用fragment。

在一个架构级别，你的APP应该有一个顶级的activity来包含绝大部分业务相关的fragment。你也可能还有一些辅助

的activity ，这些辅助的activity与主activity 通信很简单限制在这两种方法 `Intent.setData()` 或 `Intent.setAction()` 或类似的方法。

Java 包结构

Android 应用程序在架构上大致是Java中的Model-View-Controller结构。 在Android 中 Fragment和Activity通常上是控制器类(http://www.informit.com/articles/article.aspx?p=2126865). 换句话说，他们是用户接口的部分，同样也是Views视图的部分。

正是因为如此，才很难严格的将fragments (或者 activities) 严格的划分成 控制器controlloers还是视图 views。 最还是将它们放在自己单独的 `fragments` 包中。只要你遵循之前提到的建议，Activities 则可以放在顶级目录下。 若果你规划有2到3个以上的activity，那么还是同样新建一个 `activities` 包吧。

然而，这种架构可以看做是另一种形式的MVC， 包含要被解析API响应的JSON数据，来填充的POJO的 `models` 包中。 和一个 `views` 包来包含你的自定义视图、通知、导航视图，widgets等等。 适配器Adapter是在数据和视图之间。然而他们通常需要通过 `getView()` 方法来导出一些视图， 所以你可以将 `adapters` 包放在 `views` 包里面。

一些控制器角色的类是应用程序级别的，同时是接近系统的。 这些类放在 `managers` 包下面。 一些繁杂的数据处理类，比如说"DateUtils",放在 `utils` 包下面。 与后端交互负责网络处理类，放在 `network` 包下面。

总而言之，以最接近用户而不是最接近后端去安排他们。

```
com.futurice.project
├─ network
├─ models
├─ managers
├─ utils
├─ fragments
└─ views
   ├─ adapters
   ├─ actionbar
   ├─ widgets
   └─ notifications
```

资源文件 Resources

- **命名** 遵循前缀表明类型的习惯，形如 `type_foo_bar.xml` 。例如： `fragment_contact_details.xml` , `view_primary_button.xml` , `activity_main.xml` .

组织布局文件 若果你不确定如何排版一个布局文件，遵循一下规则可能会有帮助。

- 每一个属性一行，缩进4个空格
- `android:id` 总是作为第一个属性
- `android:layout_****` 属性在上边
- `style` 属性在底部
- 关闭标签 `/>` 单独起一行，有助于调整和添加新的属性
- 考虑使用Designtime attributes 设计时布局属性，Android Studio已经提供支持，而不是硬编码 `android:text`

(译者注：墙内也可以参考stormzhang的这篇博客链接)。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   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"
   android:orientation="vertical"
   >

   <TextView
      android:id="@+id/name"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
      android:text="@string/name"
      style="@style/FancyText"
      />

   <include layout="@layout/reusable_part" />

</LinearLayout>
```

作为一个经验法则, `android:layout_****` 属性应该在 layout XML 中定义,同时其它属性 `android:****` 应放在 styler XML中。此规则也有例外，不过大体工作 的很好。这个思想整体是保持layout属性(positioning, margin, sizing) 和content属性在布局文件中，同时将所有的外观细节属性（colors, padding, font）放 在style文件中。

例外有以下这些:

- `android:id` 明显应该在layout文件中
- layout文件中 `android:orientation` 对于一个 `LinearLayout` 布局通常更有意义
- `android:text` 由于是定义内容，应该放在layout文件中
- 有时候将 `android:layout_width` 和 `android:layout_height` 属性放到一个style中作为一个通用的风格中更有意义，但是默认情况下这些应该放到layout文件中。

使用styles 几乎每个项目都需要适当的使用style文件，因为对于一个视图来说有一个重复的外观是很常见的。 在应用中对于大多数文本内容，最起码你应该有一个通用的style文件，例如：

```
<style name="ContentText">
   <item name="android:textSize">@dimen/font_normal</item>
   <item name="android:textColor">@color/basic_black</item>
</style>
```

应用到TextView 中:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/price"
   style="@style/ContentText"
   />
```

你或许需要为按钮控件做同样的事情，不要停止在那里。将一组相关的和重复`android:****`的属性放到一个通用的style中。

将一个大的style文件分割成多个文件 你可以有多个`styles.xml`文件。Android SDK支持其它文件，`styles`这个文件名称并没有作用，起作用的是在文件 里xml的``标签。因此你可以有多个style文件`styles.xml`，`style_home.xml`，`style_item_details.xml`，`styles_forms.xml`。 不用于资源文件路径需要为系统构建起的有意义，在`res/values`目录下的文件可以任意命名。

`colors.xml`是一个调色板 在你的`colors.xml`文件中应该只是映射颜色的名称一个RGBA值，而没有其它的。不要使用它为不同的按钮来定义RGBA值。

不要这样做

```
<resources>
   <color name="button_foreground">#FFFFFF</color>
   <color name="button_background">#2A91BD</color>
   <color name="comment_background_inactive">#5F5F5F</color>
   <color name="comment_background_active">#939393</color>
   <color name="comment_foreground">#FFFFFF</color>
   <color name="comment_foreground_important">#FF9D2F</color>
   ...
   <color name="comment_shadow">#323232</color>
```

使用这种格式，你会非常容易的开始重复定义RGBA值，这使如果需要改变基本色变的很复杂。同时，这些定义是跟一些环境关联起来的，如`button`或者`comment`，应该放到一个按钮风格中，而不是在`color.xml`文件中。

相反，这样做:

```
<resources>

   <!-- grayscale -->
   <color name="white"     >#FFFFFF</color>
   <color name="gray_light">#DBDBDB</color>
   <color name="gray"      >#939393</color>
   <color name="gray_dark" >#5F5F5F</color>
   <color name="black"     >#323232</color>

   <!-- basic colors -->
   <color name="green">#27D34D</color>
   <color name="blue">#2A91BD</color>
   <color name="orange">#FF9D2F</color>
   <color name="red">#FF432F</color>

</resources>
```

向应用设计者那里要这个调色板，名称不需要跟"green", "blue", 等等相同。 "brand_primary", "brand_secondary", "brand_negative" 这样的名字也是完全可以接受的。 像这样规范的颜色很容易修改或重构，会使应用一共使用了多少种不同的颜色变得非常清晰。 通常一个具有审美价值的UI来说，减少使用颜色的种类是非常重要的。

像对待colors.xml一样对待dimens.xml文件 与定义颜色调色板一样，你同时也应该定义一个空隙间隔和字体大小的“调色板”。 一个好的例子，如下所示：

```
<resources>

    <!-- font sizes -->
    <dimen name="font_larger">22sp</dimen>
    <dimen name="font_large">18sp</dimen>
    <dimen name="font_normal">15sp</dimen>
    <dimen name="font_small">12sp</dimen>

    <!-- typical spacing between two views -->
    <dimen name="spacing_huge">40dp</dimen>
    <dimen name="spacing_large">24dp</dimen>
    <dimen name="spacing_normal">14dp</dimen>
    <dimen name="spacing_small">10dp</dimen>
    <dimen name="spacing_tiny">4dp</dimen>

    <!-- typical sizes of views -->
    <dimen name="button_height_tall">60dp</dimen>
    <dimen name="button_height_normal">40dp</dimen>
    <dimen name="button_height_short">32dp</dimen>

</resources>
```

布局时在写 margins 和 paddings 时，你应该使用`spacing_****`尺寸格式来布局，而不是像对待String字符串一样直接写值。 这样写会非常有感觉，会使组织和改变风格或布局是非常容易。

避免深层次的视图结构 有时候为了摆放一个视图，你可能尝试添加另一个LinearLayout。你可能使用这种方法解决：

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   >

   <RelativeLayout
      ...
      >

      <LinearLayout
         ...
         >

         <LinearLayout
            ...
            >

            <LinearLayout
               ...
               >
            </LinearLayout>

         </LinearLayout>

      </LinearLayout>

   </RelativeLayout>

</LinearLayout>
```

即使你没有非常明确的在一个layout布局文件中这样使用，如果你在Java文件中从一个view inflate（这个inflate翻译不过去，大家理解就行） 到其他views当中，也是可能会发生的。

可能会导致一系列的问题。你可能会遇到性能问题，因为处理起需要处理一个复杂的UI树结构。 还可能会导致以下更严重的问题StackOverflowError.

因此尽量保持你的视图tree：学习如何使用RelativeLayout, 如何 optimize 你的布局 和如何使用 `` 标签.

小心关于WebViews的问题. 如果你必须显示一个web视图， 比如说对于一个新闻文章，避免做客户端处理HTML的工作， 最好让后端工程师协助，让他返回一个 "*纯*" HTML。 WebViews 也能导致内存泄露 当保持引他们的Activity，而不是被绑定到ApplicationContext中的时候。 当使用简单的文字或按钮时，避免使用WebView，这时使用TextView或Buttons更好。

测试框架

Android SDK的测试框架还处于初级阶段，特别是关于UI测试方面。Android Gradle 目前实现了一个叫 `connectedAndroidTest` 的测试， 它使用一个JUnit 为Android提供的扩展插件 extension of JUnit with helpers for Android.可以跑你生成的JUnit测试，

只当做单元测试时使用 Robolectric ，views 不用 它是一个最求提供"不连接设备的"为了加速开发的测试， 非常时候做 models 和 view models 的单元测试。 然而，使用Robolectric测试时不精确的，也不完全对UI测试。 当你对有关动画的UI元素、对话框等，测试时会有问题， 这主要是因为你是在 “在黑暗中工作” （在没有可控的界面情况下测试）

*_Robotium 使写UI测试非常简单。 *_ 对于UI测试你不需 Robotium 跑与设备连接的测试。 但它可能会对你有益，是因为它有许多来帮助类的获得和分析视图，控制屏幕。 测试用例看起来像这样简单：

```
solo.sendKey(Solo.MENU);
solo.clickOnText("More"); // searches for the first occurence of "More" and clicks on it
solo.clickOnText("Preferences");
solo.clickOnText("Edit File Extensions");
Assert.assertTrue(solo.searchText("rtf"));
```

模拟器

如果你全职开发Android App,那么买一个Genymotion emulatorlicense吧。 Genymotion 模拟器运行更快的秒帧的速度，比起典型的AVD模拟器。他有演示你APP的工具，高质量的模拟网络连接，GPS位置，等等。它同时还有理想的连接测试。 你若涉及适配使用很多不同的设备，买一个Genymotion 版权是比你买很多真设备便宜多的。

注意：Genymotion模拟器没有装载所有的Google服务，如Google Play Store和Maps。你也可能需 要测试Samsung指定的API，若这样的话你还是需要购买一个真实的Samsung设备。

混淆配置

ProGuard 是一个在Android项目中广泛使用的压缩和混淆打包的源码的工具。

你是否使用ProGuard取决你项目的配置，当你构建一个release版本的apk时，通常你应该配置gradle文件。

```
buildTypes {
    debug {
        minifyEnabled false
    }
    release {
        signingConfig signingConfigs.release
        minifyEnabled true
        proguardFiles 'proguard-rules.pro'
    }
}
```

为了决定哪些代码应该被保留，哪些代码应该被混淆，你不得不指定一个或多个实体类在你的代码中。 这些实体应该是指定的类包含main方法，applets，midlets，activities，等等。 Android framework 使用一个默认的配置文件，可以在 SDK_HOME/tools/proguard/proguard-android.txt 目录下找到。自定义的工程指定的 project-specific 混淆规则，如在 my-project/app/proguard-rules.pro 中定义， 会被添加到默认的配置中。

关于 ProGuard 一个普遍的问题，是看应用程序是否崩溃并报 ClassNotFoundException 或者

`NoSuchFieldException` 或类似的异常， 即使编译是没有警告并运行成功。 这意味着以下两种可能：

1. ProGuard 已经移除了类，枚举，方法，成员变量或注解，考虑是否是必要的。
2. ProGuard 混淆了类，枚举，成员变量的名称，但是这些名字又被拿原始名称使用了，比如通过Java的反射。

检查 `app/build/outputs/proguard/release/usage.txt` 文件看有问题的对象是否被移除了。 检查 `app/build/outputs/proguard/release/mapping.txt` 文件看有问题的对象是否被混淆了。

In order to prevent ProGuard from *stripping away* needed classes or class members, add a `keep` options to your proguard config: 以防 ProGuard *剥离* 需要的类和类成员，添加一个 `keep` 选项在你的 proguard 配置文件中：

```
-keep class com.futurice.project.MyClass { *; }
```

防止 ProGuard *混淆* 一些类和成员，添加 `keepnames`：

```
-keepnames class com.futurice.project.MyClass { *; }
```

查看this template's ProGuard config 中的一些例子。 更多例子请参考Proguard。

在构建项目之初，发布一个版本 来检查ProGuard规则是否正确的保持了重要的部分。 同时无论何时你添加了新的类库，做一个发布版本，同时apk在设备上跑起来测试一下。 不要等到你的app要发布 "1.0"版本了才做版本发布，那时候你可能会碰到好多意想不到的异常，需要一些时间去修复他们。

Tips每次发布新版本都要写 `mapping.txt`。每发布一个版本，如果用户遇到一个bug，同时提交了一个混淆过的堆栈跟踪。 通过保留 `mapping.txt` 文件，来确定你可以调试的问题。

DexGuard 若果你需要核心工具来优化，和专门混淆的发布代码，考虑使用DexGuard, 一个商业软件，ProGuard也是有他们团队开发的。 它会很容易将Dex文件分割，来解决65K个方法限制问题。

致谢

感谢Antti Lammi, Joni Karppinen, Peter Tackage, Timo Tuominen, Vera Izrailit, Vihtori Mäntylä, Mark Voit, Andre Medeiros, Paul Houghton 这些人和Futurice 开发者分享他们的Android开发经验。

License

Translation

Translated to Chinese by andyiac