Gradle for Android 第二篇(Build.gradle入门)


neu
3.1k • 1 • 5

发布于
2016-01-01

这一系列暂不更新，相关技术讨论，请移步微信群，谢谢，希望大家多多支持！

新年新气象，奋斗的一年，在这一章，我们将学习以下内容：

- 理解Gradle文件
- 编写简单的构建任务
- 自制构建脚本

如果你还没有看grade for android系列的第一篇博客，请先查看：

Gradle for Android 第一篇(从 Gradle 和 AS 开始)

Gradle for Android 第三篇(依赖管理)

Gradle for Android 第四篇(构建变体)

Gradle for Android 第五篇(多模块构建)

Gradle for Android 第六篇(测试)

Gradle for Android 第七篇(Groovy入门)

理解Gradle脚本

当然我们现在讨论的所有内容都是基于Android studio的，所以请先行下载相关工具。当我们创建一个新的工程，Android studio会默认为我们创建三个gradle文件，两个build.gradle，一个settings.gradle，build.gradle分别放在了根目录和moudle目录下，下面是gradle文件的构成图：

```
MyApp
   ├── build.gradle
   ├── settings.gradle
   └── app
       └── build.gradle
```

setting.gradle解析

当你的app只有一个模块的时候，你的setting.gradle将会是这样子的：

```
include ':app'
```

setting.gradle文件将会在初始化时期执行，关于初始化时期，可以查看上一篇博客，并且定义了哪一个模块将会被构建。举个例子，上述setting.gradle包含了app模块，setting.gradle是针对多模块操作的，所以单独的模块工程完全可以删除掉该文件。在这之后，Gradle会为我们创建一个Setting对象，并为其包含必要的方法，你不必知道Settings类的详细细节，但是你最好能够知道这个概念。

根目录的build.gradle

该gradle文件是定义在这个工程下的所有模块的公共属性，它默认包含二个方法：

```
buildscript {
     repositories {
         jcenter()
     }
      dependencies {
          classpath 'com.android.tools.build:gradle:1.2.3'
      }
}
allprojects {
     repositories {
          jcenter()
     }
}
```

buildscript方法是定义了全局的相关属性，repositories定义了jcenter作为仓库。一个仓库代表着你的依赖包的来源，例如maven仓库。dependencies用来定义构建过程。这意味着你不应该在该方法体内定义子模块的依赖包，你仅仅需要定义默认的Android插件就可以了，因为该插件可以让你执行相关Android的tasks。

allprojects方法可以用来定义各个模块的默认属性，你可以不仅仅局限于默认的配置，未来你可以自己创造tasks在allprojects方法体内，这些tasks将会在所有模块中可见。

模块内的build.gradle

模块内的gradle文件只对该模块起作用，而且其可以重写任何的参数来自于根目录下的gradle文件。该模块文件应该是这样：

```
apply plugin: 'com.android.application'
  android {
      compileSdkVersion 22
      buildToolsVersion "22.0.1"
      defaultConfig {
          applicationId "com.gradleforandroid.gettingstarted"
          minSdkVersion 14
          targetSdkVersion 22
          versionCode 1
          versionName "1.0"
      }
      buildTypes {
          release {
              minifyEnabled false
              proguardFiles getDefaultProguardFile
               ('proguard-android.txt'), 'proguard-rules.pro'
          }
       }
   }
   dependencies {
      compile fileTree(dir: 'libs', include: ['*.jar'])
      compile 'com.android.support:appcompat-v7:22.2.0'
    }
```

插件

该文件的第一行是Android应用插件，该插件我们在上一篇博客已经介绍过，其是google的Android开发团队编写的插件，能够提供所有关于Android应用和依赖库的构建，打包和测试。

Android

该方法包含了所有的Android属性，而唯一必须得属性为compileSdkVersion和buildToolsVersion：

- compileSdkVersion：编译该app时候，你想使用到的api版本。
- buildToolsVersion：构建工具的版本号。

构建工具包含了很多实用的命令行命令，例如aapt,zipalign,dx等，这些命令能够被用来产生多种多样的应用程序。你可以通过sdk manager来下载这些构建工具。

defaultConfig方法包含了该app的核心属性，该属性会重写在AndroidManifest.xml中的对应属性。

```
defaultConfig {
        applicationId "com.gradleforandroid.gettingstarted"
        minSdkVersion 14
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
}
```

> 第一个属性是applicationId，该属性复写了AndroidManifest文件中的包名package name，但是关于applicationId和package name有一些不同。在gradle被用来作为Android构建工具之前，package name在AndroidManifest.xml有两个作用：其作为一个app的唯一标示，并且其被用在了R资源文件的包名。

Gradle能够很轻松的构建不同版本的app,使用构建变种。举个例子，其能够很轻松的创建一个免费版本和付费版本的app。这两个版本需要分隔的标示码，所以他们能够以不同的app出现在各大应用商店，当然他们也能够同时安装在一个手机中。资源代码和R文件必须拥有相同的包名，否则你的资源代码将需要改变，这就是为什么Android开发团队要将package name的两大功能拆分开。在AndroidManifest文件中定义的package name依然被用来作为包名和R文件的包名。而applicationid将被用在设备和各大应用商店中作为唯一的标示。

接下来将是minSdkVersion和targetSdkVersion。这两个和AndroidManifest中的<uses-sdk>很像。minSdkVersion定义为最小支持api。

versionCode将会作为版本号标示，而versionName毫无作用。

所有的属性都是重写了AndroidManifest文件中的属性，所以你没必要在AndroidManifest中定义这些属性了。

buildTypes方法定义了如何构建不同版本的app，我们将在下一篇博客中有所介绍。

依赖包

依赖模块作为gradle默认的属性之一（这也是为什么其放在了Android的外面），为你的app定义了所有的依赖包。默认情况下，我们依赖了所有在libs文件下的jar文件，同时包含了AppCompat这个aar文件。我们将会在下一篇博客中讨论依赖的问题。

让我们开始tasks吧

如果你想知道你多少tasks可以用，直接运行gradlew tasks，其会为你展示所有可用的tasks。当你创建了一个Android工程，那么将包含Android tasks，build tasks，build setup tasks，help tasks，install tasks，verification tasks等。

基本的tasks

android插件依赖于Java插件，而Java插件依赖于base插件。

base插件有基本的tasks生命周期和一些通用的属性。

base插件定义了例如assemble和clean任务，Java插件定义了check和build任务，这两个任务不在base插件中定义。

这些tasks的约定含义：

- assemble: 集合所有的output
- clean: 清除所有的output
- check: 执行所有的checks检查，通常是unit测试和instrumentation测试
- build: 执行所有的assemble和check

Java插件同时也添加了source sets的概念。

Android tasks

android插件继承了这些基本tasks,并且实现了他们自己的行为：

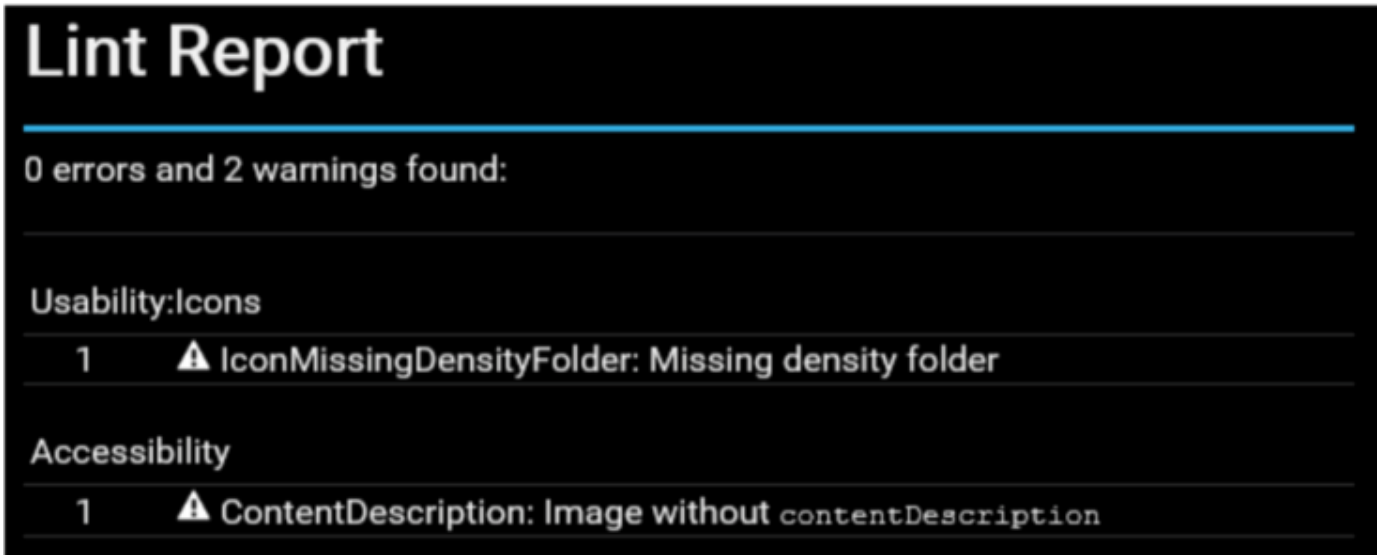
- assemble 针对每个版本创建一个apk
- clean 删除所有的构建任务，包含apk文件
- check 执行Lint检查并且能够在Lint检测到错误后停止执行脚本
- build 执行assemble和check

默认情况下assemble tasks定义了assembleDebug和assembleRelease，当然你还可以定义更多构建版本。除了这些tasks,android 插件也提供了一些新的tasks:

- connectedCheck 在测试机上执行所有测试任务
- deviceCheck 执行所有的测试在远程设备上

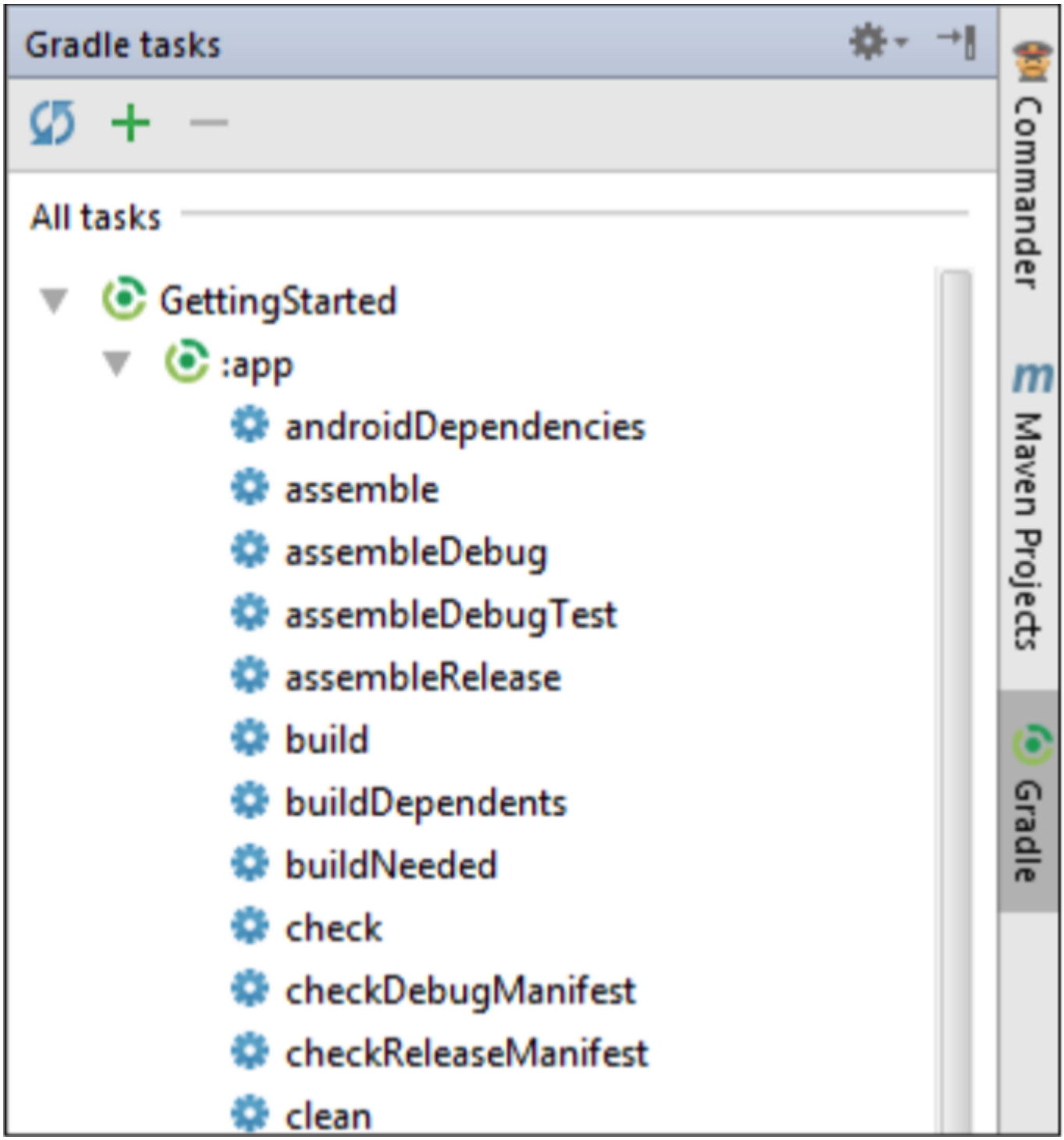
- installDebug和installRelease 在设备上安装一个特殊的版本
- 所有的install task对应有uninstall 任务

build task依赖于check任务，但是不依赖于connectedCheck或者deviceCheck，执行check任务的使用Lint会产生一些相关文件，这些报告可以在app/build/outputs中查看：

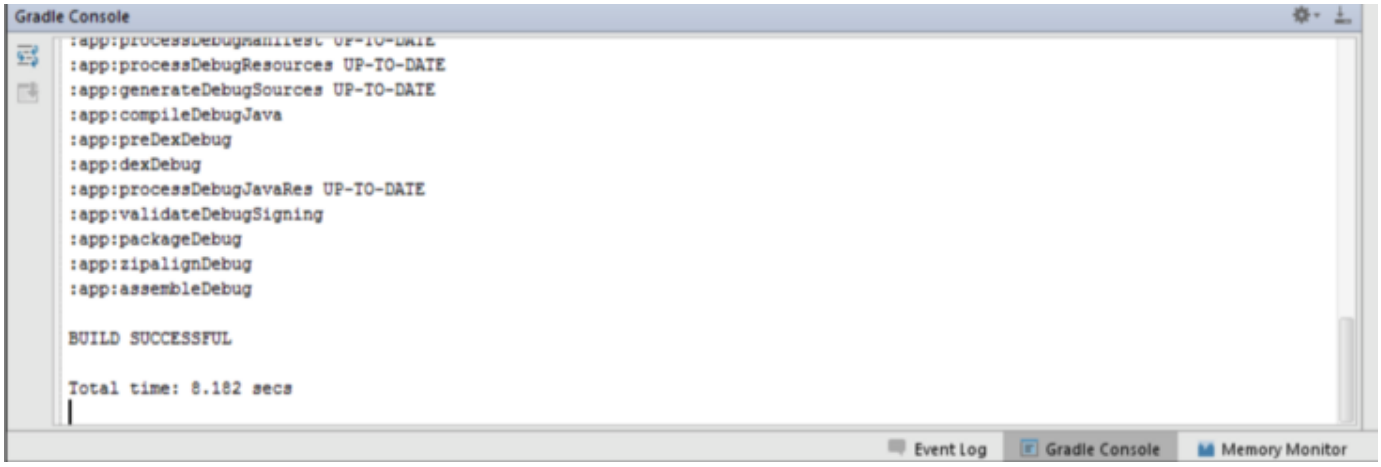


android studio的tasks

你根本不必要去执行gradle脚本在命令行中，Android studio有其对应的工具：

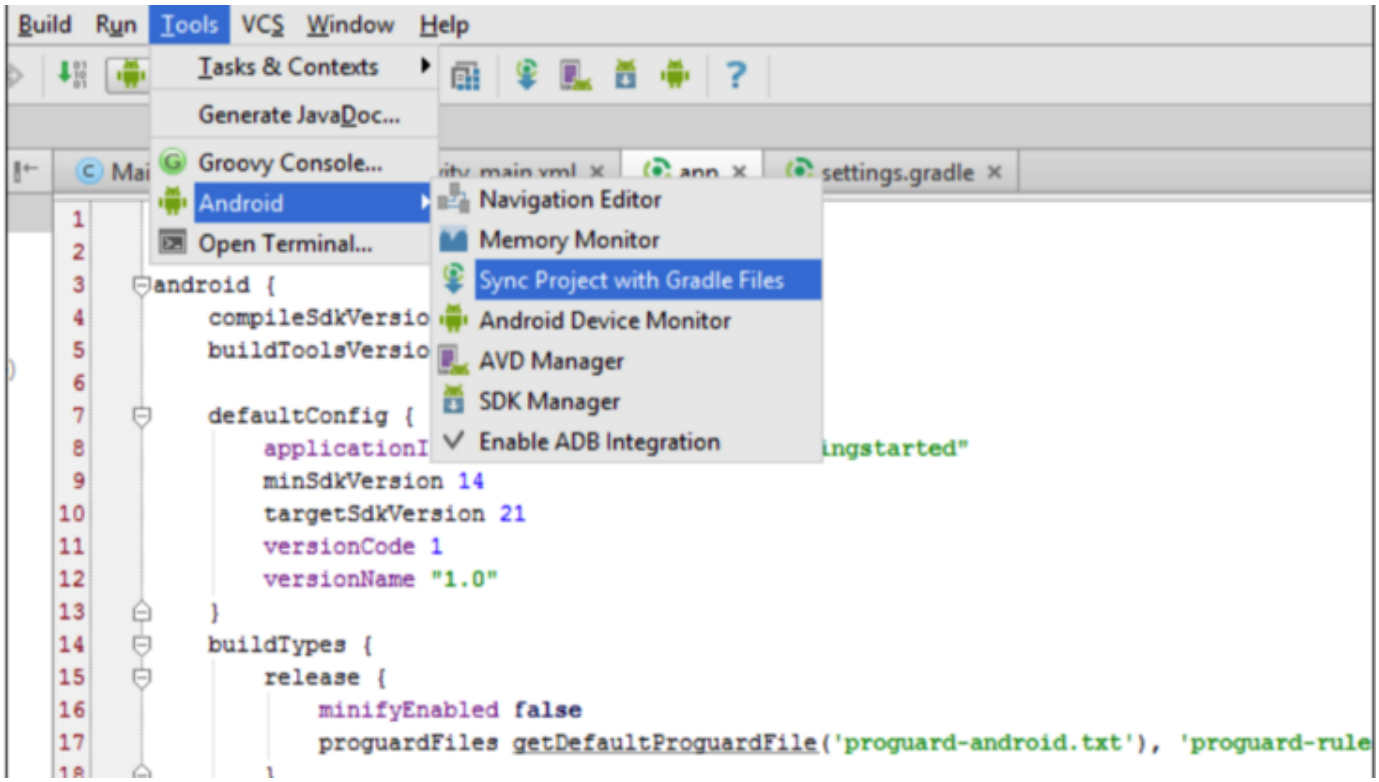


在这个界面，你要做的就是双击了。当然你也可以在Android studio中打开命令行，执行相关命令，具体操作就不介绍了。



自定义构建

当你在Android studio中自定义了gradle文件，需要更新project:



其实该按钮，执行了generateDebugSources tasks，该任务会生成所有必要的classes文件。

BuildConfig和resources

```
android {
    buildTypes {
        debug {
            buildConfigField "String", "API_URL",
               "\"http://test.example.com/api\""
               buildConfigField "boolean", "LOG_HTTP_CALLS", "true"
     }
       release {
            buildConfigField "String", "API_URL",
                "\"http://example.com/api\""
               buildConfigField "boolean", "LOG_HTTP_CALLS","false"
     }
 }
```

类似这些定义的常量，当定义了这些属性后，你完全可以在代码中使用：BuildConfig.API_URL和BuildConfig.LOG_HTTP

最近，Android tools team也让其里面定义string变为可能：

```
android {
       buildTypes {
           debug {
               resValue "string", "app_name", "Example DEBUG"
           }
           release {
               resValue "string", "app_name", "Example"
            }
       }
}
```

你可以在代码中使用这些string。其中""不是必须得。

全局设置

如果你有很多模块在一个工程下，你可以这么定义你的project文件。

```
allprojects {
       apply plugin: 'com.android.application'
       android {
           compileSdkVersion 22
           buildToolsVersion "22.0.1"
       }
 }
```

这只会在你的所有模块都是Android app应用的时候有效。你需要添加Android 插件才能访问Android的tasks。更好的做法是你在全局的gradle文件中定义一些属性，然后再模块中运用它们。比如你可以在根目录下这么定义：

```
 ext {
       compileSdkVersion = 22
       buildToolsVersion = "22.0.1"
}
```

那么你在子模块中就可以使用这些属性了：

```
android {
       compileSdkVersion rootProject.ext.compileSdkVersion
       buildToolsVersion rootProject.ext.buildToolsVersion
```

```
}
```

Project properties文件

上述方法是一种办法，当然还有很多办法：

- ext方法
- gradle.properties文件
- -p参数

```
ext {
     local = 'Hello from build.gradle'
}
   task printProperties << {
     println local        // Local extra property
     println propertiesFile        // Property from file
     if (project.hasProperty('cmd')) {
       println cmd        // Command line property
     }
}
```

当然你可以在gradle.properties中定义：

```
propertiesFile = Hello from gradle.properties
```

你也可以输入命令行：

```
$ gradlew printProperties -Pcmd='Hello from the command line'
:printProperties
Hello from build.gradle
Hello from gradle.properties
Hello from the command line
```

总结

在这篇博客中，我们细致的查看了Android studio生成的三个gradle文件，现在你应该能够自己去创建自己的gradle文件，我们还学习了最基本的构建任务，学习了Android 插件以及其tasks。

在接下来的几年里，Android开发生态将会爆炸性增长，很多有趣的依赖库将会让每个人去使用，在下一篇博客里面，我们将看看我们能有几种方式添加我们的依赖库，这样我们才能够避免造轮子。

android android-studio gradle

阅读 53.1k • 更新于 2016-03-04

neu

最新android特性学习，翻译AndroidHive，google develop大神博客