Android N预览版的开发者注意：Instant Run目前与Jack toolchain不兼容。新的toolchain需要使用Java 8的语言特性来支持Android N。了解更多，阅读[new Java 8 language features and Jack](http://developer.android.com/preview/j8-jack.html)。

自Android Studio2.0引入的Instant Run 显著地减少了**Run** http://tools.android.com/_/rsrc/1458888244637/tech-docs/instant-run/as-run.png 和**Debug** http://tools.android.com/_/rsrc/1458888261919/tech-docs/instant-run/as-debugbutton.png 在部署app时的时间。不用重新编译一次app就能推送你做出的改动到app,这样代码的修改看起来就像及时的一样。

Instant Run的使用条件:

* 编译调试版本
* 使用的Android Gradle插件版本在2.0.0或更高
* Android版本4.0（API15）或更高

部署完毕后在Run（或者Debug）按钮的图标中会出现一个黄色的小闪电图标，这标识

Instant Run已经可以使用了，当你下一次使用这两个命令的时候，Android Studio 会推送你所做的更改，在一些情况下甚至不用重启app你能看到新代码的效果。

**注意**：Instant Run会暂时禁用JaCoCo和混淆。由于Instant Run只会影响Debug版本，这对于Release版的JaCoCo和混淆并不会影响

**工作原理**：执行 热拔插（hot swap）, warm swap*（注释1）*, 或者 冷抽换cold swap。

Instant Run会自动地根据你所做出的改变去选择swap模式。

下边的表格描述了Instant Run在这些改动推送到目标设备时的行为。（翻译比较难理解，一句话就是【这玩意会自己判断啥时候即时运行，啥时候重新编译或者需要重启app】）

|  |  |
| --- | --- |
| 所做的改动 | Instant Run的对应行为 |
| 改变实现一个已经存在的实例方法或者静态方法 | 支持Hot Swap：这是最快的Swap类型，改动几乎即时生效。App保存运行状态，此时会生成一个重构的存根方法，用到的时候就会调用这个方法 |
| 修改或者删除一个已经存在res资源 | 支持Warm Swap：仍然算快的Swap类型，但当Instant Run推送所改变的资源到目标设备的时候会重启当前activity。App保持运行状态，但是可能屏幕会有一些小闪烁随着Activity重启。正常的 |
| 结构上的代码改变，例如：   * 添加、删除或修改   + 一个注解   + 一个实例字段   + 一个静态字段   + 一个静态方法签名   + 一个实例方法签名 * 当前类的父类的改动 * 实现的接口列表改动 * 改变一个类的静态初始化方法 * 利用动态的资源ID改版布局元素顺序 | API 21或更高  支持Cold Swap：Instant Run推送这个结构级别修改过的代码到目标设备，然后重启app。  API 20或更低  重新编译Apk，重新部署 |
| * 修改清单（manifest）配置 * 修改了清单引用的资源 * 修改Android小控件UI（需要清理项目再重新运行） | \* 当清单文件被修改或者修改了清单文件引用的资源时，Android Studio会自动部署一个新的构建以便于把改动推送到设备上。因为app的某些信息例如：app名字、app图标以及意图过滤器，都是在APK安装到设备时由清单文件控制的。  \* 如果你的编译进程自动更新清单文件配置，例如机械的重复的versionCode或者versionName，Instant Run的全部性能提升将会有所限制。我们建议你在调试版本中禁用自动更新清单文件的配置 |

Android Studio默认设置为执行了Hot Swap时重启当前activity（app保存运行）。禁用此设置可以按照如下操作：

1. 打开设置或者配置对话框
2. 找到**Build, Execution, Deployment** > **Instant Run**选项
3. 取消**Restart activity on code changes**选项旁的勾选框

当activity自动重启被禁用时，你可以通过菜单栏 **Run** > **Restart Activity**手动重启当前activity

**Rerun**

当有例如 [onCreate()](http://adarshf-linux.kir.corp.google.com:9000/reference/android/app/Application.html#onCreate())等方法的代码修改时，你就需要重启app来使所做的改变生效了。你可以点击**Rerun**停止app，部署并重启新修改的app。

如果你需要clean build项目，使用主菜单中的**Run** > **Clean and Rerun 'app'**选项或者按下**Shift**并点击**Rerun** C:\Users\spark\AppData\Local\Temp\msohtmlclip1\02\clip_image001.png，此时app会停止运行，同时执行完整的clean build，完成后会把apk部署到你的设备上面。

### 优化项目配置以适应Instant Run

AS会为使用Gradle 2.0.0 或者更高的项目自动启用Instant Run.

(如果需要)升级现有项目的Gradle插件到最新版本按如下操作：

1. 打开设置对话框：
2. 找到**Build, Execution, Deployment** > **Instant Run**选项点击 **Update Project**，如图3所示，如果 **Update Project**不可用，说明项目的该Gradle插件已经是最新版本了

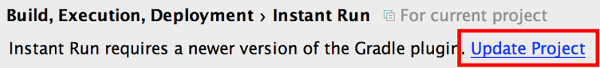


图3.更新已有项目的Gradle插件

当然，你同样需要[设置为调试编译模式](https://developer.android.com/studio/run/index.html#changing-variant)才能使用Instant Run

#### 通过配置DEX资源改善编译需要的时间

当你执行cleanb build时，AS通过一些手段来运行Instant Run推送更新的代码或者资源到你的app。虽然这样更新代码比之前快得多了，但是第一次编译耗时更长。综上，你可以通过一些[DexOptions](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html)设置来改善编译进程：

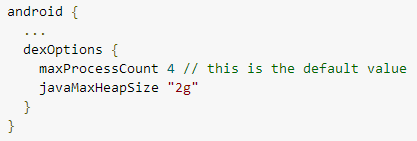
* [maxProcessCount](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:maxProcessCount)（最大进程数）：

调整Dex processes的最大并发进程数。如果Gradle的伺服系统已经在运行了，你要先将它停止才能让这个新最大并发进程数生效。你可以通过下列命令行停止Gradle的伺服系统：

* + Windows：gradlew –stop
  + Linux/Mac OSX：./gradlew –stop
* [javaMaxHeapSize](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:javaMaxHeapSize)(最大内存)：

调整dex operation的内存池的最大值，输入值的时候，可以输入小写字母“k”标识千字节，”m“、”g“表示兆字节、一亿字节

下面实例了在module的build.gradle文件中将最大进程数设置为4，最大内存为2g



你应该尝试设置增加这个值来看看你编译时的时间变化。但是如果将太多的资源分配给了dexing进程，可能会让性能降低。

#### 开启同步Dexing和Java增量编译

Gradle2.1.0或更高版本对编译进程进行了以下优化：Java增量编译和Dexing同步化。

Java增量编译默认就是开启的，通过只编译作出修改的部分或者需要编译的部分来减少编译时间。

Dexing同步化，在编译进程内执行dexing而不是在额外的VM金成中单独执行。这不仅仅让增量编译快很多，更是大大地提升了完整的编译的速度。启用这个功能，你需要将Gradle的伺服系统（daemon）的最大堆内存至少为2048MB。你可以通过在项目的gradle.properties文件中添加如下指令来完成这个设置。



如果你在module的build.gradle文件中设置过最大内存的值（[javaMaxHeapSize](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:javaMaxHeapSize)），你需要在gradle.properties文件中将最大内存值设置为[javaMaxHeapSize](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:javaMaxHeapSize)+1024MB。例如，你已经将[javaMaxHeapSize](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:javaMaxHeapSize)设置为“2g”，你就需要在gradle.properties文件中添加如下指令：



#### 将项目添加进Windows Defender的白名单

在Windows系统中，当使用Instant Run的时候Windows Defender可能会造成（系统）缓慢。如果你使用了Windows Defender，请将你项目所在的文件夹[添加到Windows Defender恶意软件（malware scans）扫描的白名单](http://answers.microsoft.com/en-us/protect/wiki/protect_defender-protect_scanning/how-to-exclude-a-filefolder-from-windows-defender/f32ee18f-a012-4f02-8611-0737570e8eee)。

#### 在调试版本中禁用Crashlytics（奔溃统计分析）

显然Crashlytics会增加编译时间，在开发app的过程中为了提高编译性能可以[禁用Crashlytics](https://docs.fabric.io/android/crashlytics/build-tools.html#disabling-crashlytics-for-debug-builds)。

### Instant Run的一些限制

Instant Run的设计目的就是为了在大多数情况下提升编译和部署的速度，然而有些情况可能会影响它的表现甚至是app的兼容性。当遇到除一下情况外的Instant Run方面问题时，你可以[提交bug](http://tools.android.com/filing-bugs)。

#### 部署到多设备

Instant Run针对不同API版本的设备使用了不同的技术来实现Hot、Warm、Clod Swap。当一次性部署app到多设备的时候，AS会暂时关闭Instant Run（译者注：原文档未说明时候会自动开启Instant Run）。

#### 使用了multidex（分包）

当你的项目使用的是老版的分包(就是说在build.gradle中有设置multiDexEnabled true并且minSdkVersion 在20及以下)而且将要部署的目标设备API等级在20及以下，AS会禁用Instant Run。

当minSdkVersion 在21及以上的时候，Instant Run会自动为你配置分包，然而因为Instant Run只在调试模式下有效，在正式版时你还需要[手动配置分包](https://developer.android.com/tools/building/multidex.html#mdex-gradle)。

#### 运行可视化测试和性能分析

可视化测试将debug APK和test APK加载进测试设备的同一进程，并允许控制方法来重写app的普通生命周期以执行测试。当运行或者调试可视化测试的时候AS不会插入Instant Run所需要的方法，此时Instant Run关闭。

当分析app性能的时候，你应该关闭Instant Run。Instant Run 对于性能还是有那么一丢丢影响的。在使用Hot Swap的时候对于重写方法也有稍大的影响。这个性能影响会干扰性能分析工具给出的信息。同时，Hot Swap生成的存根方法也会使堆栈痕迹复杂。

#### 使用第三方插件

AS会暂时的禁用JaCoCo和混淆当使用Instant Run时。你不用担心这会影响你的正式版本，因为这只会在调试版本的时候发生。

某些第三方插件也有在执行字节码增强时对Instant Run在处理app时造成一些问题。当你遇到这个问题的时候并且又想使用Instant Run你可以选择在调试版本的时候禁用这些造成影响的第三方插件。也非常欢迎你[提交这个问题](http://tools.android.com/filing-bugs)来提高这些第三方插件的兼容性。

#### 推送更改到不同进程的app

Instant Run只会对你的app主进程做处理来完成Hot/Warm Swap。如果是推送更改到其他app进程，哪怕更改只是方法实现，资源文件也会执行Cold Swap。

#### 关闭Instant Run

关闭Instant Run 你可以做如下操作：

1. 打开设置对话框
2. 找到**Build, Execution, Deployment** > **Instant Run**
3. 取消**Enable Instant Run**旁的勾选框