آموزش برنامه نویسی اندروید
در محیط اندروید استودیو

۲۱ روش افزایش سرعت بیلد Gradle در اندروید استودیو

مدرس : سیدمهدی مطهری

www.android-studio.ir

به نام خدا

احتمالا با مشاهده تصویر بالا متوجه شده‌اید که در این مبحث می‌خواهیم راجع به چه موضوعی صحبت کنیم. بسته به میزان حجم یک پروژه اندرویدی، فرایند بیلد شدن آن بین چند ثانیه تا چندین دقیقه می‌تواند زمان بر باشد. بنابراین اجرای روش‌های افزایش سرعت بیلد Gradle در اندروید استودیو می‌تواند تا چندین برابر، مدت زمان این فرایند طولانی و خسته کننده را کاهش دهد.

بهینه سازی صحیح اندروید استودیو و بیلد سیستم آن یعنی گریدل، می‌تواند یک فرآیند چند دقیقه‌ای بیلد را به چند ثانیه کاهش دهد! پس توصیه می‌کنم این قسمت از سری مباحث آموزش برنامه نویسی اندروید را با دقت و تا انتها مطالعه کنید.

دلایل کاهش سرعت بیلد شدن پروژه اندرویدی

عموما وقتی صحبت از کندی سرعت اجرای یک نرم افزار به میان می‌آید در اولین قدم انگشت اتهام به سمت سخت افزار رایانه رفته و پایین بودن کانفیگ سخت افزاری سیستم به عنوان علت اصلی بروز مشکل بیان می‌شود.

قطعا سخت افزار نقش تعیین کننده‌ای در سرعت اجرای ابزار و نرم افزارها به عهده دارد اما این همه‌ی ماجرا نیست. بهینه سازی ابزار مورد استفاده نیز می‌تواند تا حد زیادی و در مواقعی تا چند برابر سرعت انجام عملیات را افزایش دهد بدون آنکه نیازی به ارتقاء سخت افزار و صرف هزینه‌های گزاف وجود داشته باشد.

حتی یک PC یا Laptop قدرتمند و با کانفیگ بالا هم از این قاعده مستثنی نبوده و در صورت عدم بهینه سازی ابزار مورد استفاده، هنگام رندر شدن یک پروژه گرافیکی سنگین یا بیلد شدن پروژه اندرویدی در اندروید استودیو با کاهش بازدهی و افزایش زمان مواجه خواهیم شد.

بنابراین به عنوان یک برنامه نویس و توسعه دهنده اندروید اگر یک سیستم با مشخصات سخت افزاری بالا در اختیار داریم نباید این حس به ما القاء شود که نیازی به بهینه سازی نیست!

در این جلسه سعی می‌کنم مهمترین مواردی که در افزایش سرعت بیلد Gradle در اندروید استودیو موثر هستند را معرفی کنم.

روش‌های افزایش سرعت بیلد Gradle در اندروید استودیو

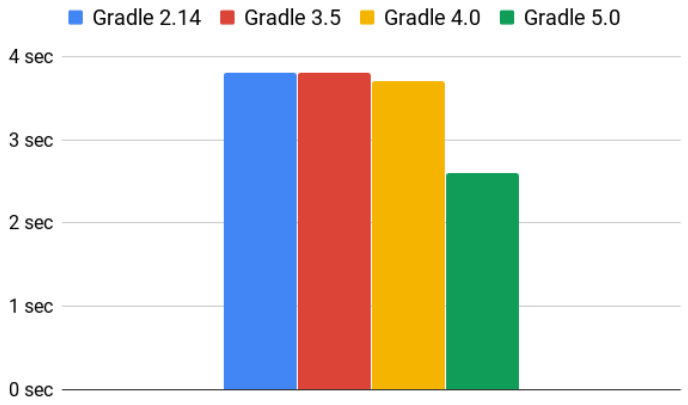
هر کدام از مواردی که در ادامه ذکر می‌شود درصد متفاوتی در کاهش نهایی زمان بیلد شدن پروژه اندرویدی سهم خواهد داشت بنابراین اکتفا کردن به یک یا چند مورد، تاثیر حداکثری را به همراه نخواهد داشت.

بنابراین درخواست می‌کنم تمامی آیتم‌ها را به دقت و با حوصله کافی مطالعه و اجرا کنید تا بتوانید حداکثر بهره وری را از سیستم خود کسب کرده و در نهایت، کمترین زمان را برای بیلد شدن پروژه‌های اندرویدی خود هدر دهید.

استفاده از جدیدترین نسخه پلاگین Gradle

به مرور زمان نسخه‌های جدیدتری از پلاگین گریدل منتشر می‌شود که تعدادی از مشکلات و باگ‌ها رفع شده و نسبت به نسخه‌های قبل بهینه تر هستند که در نتیجه مقداری از زمان انتظار ما برای بیلد شدن پروژه‌های اندرویدی کاسته می‌شود.

البته عمدتا این افزایش بازدهی به قدری ملموس نیست که بخواهد به یکباره زمان بیلد را ۵۰ درصد کاهش دهد اما بهرحال بی تاثیر هم نیست. ضمن اینکه با گذشت زمان و رشد قابلیت‌ها و امکانات ابزارهای هوشمند و بویژه سیستم عامل اندروید، حجم کتابخانه‌ها، پلاگین‌ها و کدهای پروژه اندرویدی نیز افزایش یافته و به نوعی این افزایش سرعت بیلد گریدل و حجم کدها باعث همپوشانی یکدیگر می‌شود.



بر اساس چارت فوق که در مستندات وب سایت Gradle منتشر شده، در کامپایل یک پروژه جاوایی که حاوی ۱۰۰۰ ماژول بوده، مدت زمان بیلد از حدود ۴ ثانیه در گریدل ۲.۱۴ به ۲.۵ ثانیه در گریدل ۵ کاهش یافته است. یعنی چیزی حدود ۳۰ درصد.

برای بروزرسانی Gradle نیاز به انجام کار اضافی نیست. با نصب نسخه جدید اندروید استودیو و یا بروزرسانی آن چنانچه نسخه جدیدی از گریدل و البته سازگار با آن نسخه از اندروید استودیو منتشر شده باشد به صورت خودکار دریافت و جایگزین نسخه قدیمی خواهد شد.

اما چنانچه قصد دارید بدون بروزرسانی اندروید استودیو از آخرین نسخه گریدل استفاده کنید لازم است نسخه پلاگین در build.gradle (Project) در بلاک dependencies بروز شود:

```
dependencies {
    classpath "com.android.tools.build:gradle:4.1.1"
}
```

در زمان تهیه این آموزش Gradle plugin 4.1.1 روی اندروید استودیو من فعال است که البته به من پیشنهاد می‌دهد نسخه ۴.۱.۲ را نصب کنم:

build.gradle (Test)

You can use the Project Structure dialog to view and edit your project configuration

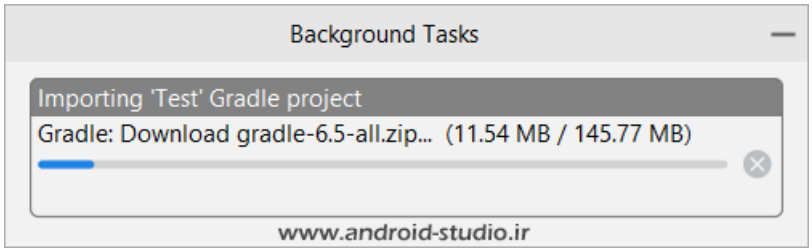
```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.1.1"
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

A newer version of com.android.tools.build:gradle than 4.1.1 is available: 4.1.2

Change to 4.1.2 Alt+Shift+Enter More actions... Alt+Enter

www.android-studio.ir

پلاگین را به ۴.۱.۲ ارتقاء داده و پروژه را Sync می‌کنم. بلافاصله اندروید استودیو شروع به دانلود گریدل ۶.۵ می‌کند:



یعنی برای استفاده از نسخه جدید پلاگین گریدل، خود گریدل نیز باید ابتدا بروز شود. لینک فایل گریدل در gradle-wrapper.properties ذخیره می‌شود که در صورت نیاز می‌توان به صورت دستی هم آنرا تغییر داد:

gradle-wrapper.properties

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-6.5-all.zip
```

تذکر: اگر در حال مطالعه این آموزش هستید یعنی تجربه کار با اندروید استودیو و مشکلات مربوط به تحریم را دارید. چنانچه از ابزاری مانند FOD برای دور زدن تحریم و فعال کردن پروکسی روی اندروید استودیو استفاده می‌کنید به دلیل اینکه این سرویس‌ها صرفا آدرس‌های تحریم شده را از پروکسی خود عبور می‌دهند و در حال حاضر سرویس Gradle بر روی IP ایران تحریم نیست، لازم است قبل از شروع دانلود، پروکسی را غیر فعال کنید.

البته شاید بعد از دریافت و نصب گریدل، اندروید استودیو بخواهد فایل‌های دیگری دریافت کند که مشمول تحریم بوده و لازم باشد مجدد پروکسی را فعال نمایید!

برای کسب اطلاع دقیق از جزئیات مربوط به نسخه‌های گریدل و پلاگین‌های سازگار با آن به صفحه Android Gradle plugin مراجعه کنید.

استفاده از آخرین نسخه Java

در زمان تهیه این آموزش Java 8 آخرین نسخه از جاوا است. جاوا ۸ (با عنوان ۱.۸) نسبت به نسخه‌های قبلی خود سرعت بیشتری دارد و اگر هنوز از جاوا ۱.۶ یا ۱.۷ استفاده می‌کنید حتما آن را به ۱.۸ یا جدیدترین نسخه موجود ارتقا دهید.

البته اگر به یاد داشته باشید در نسخه‌های ابتدایی محیط توسعه اندروید استودیو لازم بود JDK را به صورت دستی روی سیستم نصب کرده و سپس محل نصب را به آن معرفی کنیم. اما در نسخه‌های اخیر، تیم توسعه دهنده اندروید استودیو این زحمت را هم از روی دوش برنامه نویس‌های اندرویدی برداشته و یک نسخه از OpenJDK را درون IDE تعبیه کرده‌اند.

بنابراین با بروزرسانی اندروید استودیو مطمئن هستیم که آخرین نسخه از JDK را استفاده می‌کنیم. با اینحال برای اطمینان از نصب آخرین نسخه جاوا کافیست بلاک compileOptions در build.gradle (Module:app) را بررسی کنیم:

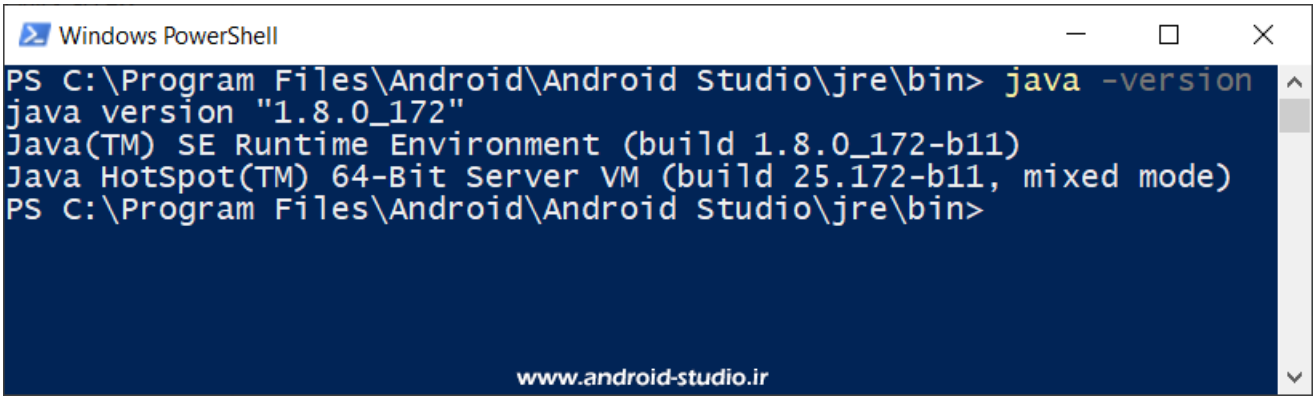
```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

در کد فوق نسخه ۸ جاوا قید شده. همچنین در محل نصب JDK توسط Command line می‌توان با دستور java -version نسخه جاوای موجود را بررسی کرد.

اگر اندروید استودیو را در مسیر پیش فرض نصب کرده‌اید محلی که می‌توان نسخه جاوا را چک کرد به صورت زیر است:

C:\Program Files\Android\Android Studio\jre\bin

در سیستم عامل ویندوز در این پنجره روی صفحه shift و راست کلیک کرده و گزینه Open PowerShell window here یا Open Command line window here را انتخاب می‌کنم تا پنجره خط فرمان باز شود:



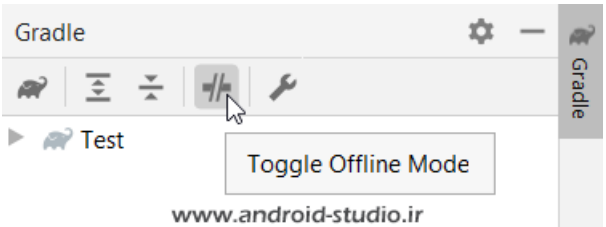
ملاحظه می‌کنید با نوشتن دستور java -version نسخه ۱.۸ نمایش داده شد.

فعال کردن حالت آفلاین Gradle

یکی دیگر از گزینه‌های افزایش سرعت بیلد Gradle در اندروید استودیو آفلاین کردن گریدل است. چنانچه گریدل در حالت پیش فرض خود یعنی Online قرار داشته باشد ممکن است در حین فرایند بیلد پروژه دانلود تعدادی فایل را آغاز کند که همین امر موجب افزایش زمان بیلد می‌شود. بهتر است فقط زمانی گریدل را در حالت آنلاین قرار دهیم که می‌خواهیم یک کتابخانه جدید به پروژه اضافه کنیم که قبلا استفاده نشده و در کش گریدل موجود نیست.

کتابخانه‌هایی که قبلا یکبار در همین پروژه یا پروژه‌ای دیگر استفاده شده و به صورت آنلاین دریافت شده‌اند تا زمانی که به صورت دستی حذف نشود در کش موجود بوده و بدون نیاز به اتصال مجدد قابل استفاده خواهد بود (مگر آنکه بخواهیم نسخه جدیدتر آن کتابخانه را نصب کنیم).

برای اینکار کافیست در نوار سمت راست اندروید استودیو و یا منوی View > Tool windows گزینه Gradle و سپس Toggle Offline Mode را انتخاب کنید:



مهاجرت به لینوکس!

قبلا هم می‌دانستم اندروید استودیو در اجرا و همچنین بیلد شدن پروژه‌ها در توزیع‌های لینوکسی محبوب مانند Ubuntu سرعت بیشتری نسبت به ویندوز دارد. اما برای اطمینان بیشتر عبارت

Does Android Studio run faster on Linux?

را گوگل کردم تا تجربه سایر افراد را در این زمینه بدانم. تقریبا همه آنها از افزایش سرعت حتی تا ۵۰% بعد از مهاجرت به Ubuntu را اعلام کرده بودند. برای نمونه می‌توانید این بحث موجود در Quara را بررسی کنید.

البته برای کسی مثل من شاید مهاجرت از ویندوز به لینوکس سخت باشد اما بهرحال یکی از گزینه‌ها بود و لازم میدانستم به آن اشاره کنم.

عدم استفاده از ورژن‌های داینامیک کتابخانه‌ها

اگر عادت کرده‌اید هنگام افزودن یک کتابخانه به پروژه اندرویدی خود برای سادگی کار بجای درج نسخه دقیق آن، یک "+" قرار دهید از امروز اشتباه خود را اصلاح کنید! داینامیک بودن نسخه کتابخانه‌ها باعث می‌شود تا گریدل هر ۲۴ ساعت یکبار به مخزن آنلاین متصل شده تا بررسی کند آیا نسخه جدیدتری منتشر شده یا نه.

برای مثال در زمان نگارش این آموزش، ورژن ۲.۹.۰ آخرین نسخه از کتابخانه Retrofit است.

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:+'
}
```

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
}
```

در کدهای فوق، مورد اول اشتباه و مورد دوم صحیح است. بگذریم از اینکه در مواردی حتی کتابخانه‌های پیش فرض پروژه هم به همین صورت تعریف شده است:

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplemen
    androidTestImplemen
}
```

Avoid using + in version numbers; can lead to unpredictable and unrepeatable builds (junit:junit:4.+)

Replace with specific version Alt+Shift+Enter More actions... Alt+Enter www.android-studio.ir

استفاده حداقلی از کتابخانه‌ها

بالا رفتن تعداد کتابخانه‌های اضافه شده به پروژه می‌تواند سرعت بیلد را به همان نسبت کاهش دهد. اگر کتابخانه‌ای را قبلا اضافه کرده‌اید و الان به هر دلیل از استفاده از آن منصرف شده‌اید حتما نسبت به حذف آن از پروژه اقدام کنید.

همچنین برخی کتابخانه‌ها ترکیبی از دو یا چند کتابخانه دیگر هستند. اگر فقط به یکی از زیر مجموعه‌های آن نیاز دارید فقط همان را اضافه کنید تا از اضافه شدن کتابخانه‌های بلا استفاده جلوگیری گردد.

ضمن اینکه توجه داشته باشید یک کتابخانه فقط یکبار به پروژه اضافه شده باشد. برای مثال کتابخانه Retrofit Gson Converter خودش از کتابخانه Gson گوگل استفاده می‌کند بنابراین نیازی نیست در کنار آن، دوباره کتابخانه Gson را تعریف کنیم.

با استفاده از دستور gradlew app:dependencies در Terminal می‌توانید لیست کامل کتابخانه‌های بکار رفته در پروژه را مشاهده کنید:

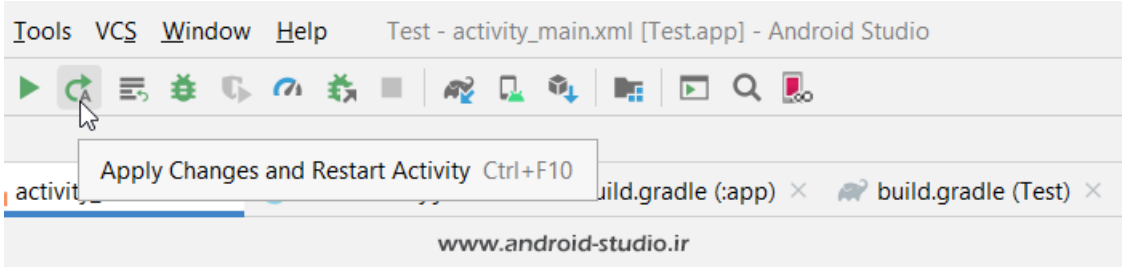
```
Terminal:  Local
+--- androidx.test.ext:junit:1.1.2
|    +--- junit:junit:4.12
|    |    \--- org.hamcrest:hamcrest-core:1.3
|    +--- androidx.test:core:1.3.0
|    |    +--- androidx.annotation:annotation:1.0.0 -> 1.1.0
|    |    +--- androidx.test:monitor:1.3.0
|    |    |    \--- androidx.annotation:annotation:1.0.0 -> 1.1.0
|    |    \--- androidx.lifecycle:lifecycle-common:2.0.0 -> 2.1.0
|    |         \--- androidx.annotation:annotation:1.1.0
|    +--- androidx.test:monitor:1.3.0 (*)
|    \--- androidx.annotation:annotation:1.0.0 -> 1.1.0
+--- androidx.test.espresso:espresso-core:3.3.0
|    +--- androidx.test:runner:1.3.0
|    |    +--- androidx.annotation:annotation:1.0.0 -> 1.1.0
|    |    +--- androidx.test:monitor:1.3.0 (*)
www.android-studio.ir
TODO  Terminal  Database Inspector  Profiler  6: Logcat  Event Log  Layout Inspector
```

نحوه کار با ترمینال در ادامه همین آموزش (ساخت پروفایل برای بیلد و بررسی آن) توضیح داده شده است.

اعمال تغییرات بجای اجرای دوباره هنگام تست و دیباگ

اگر پروژه روی شبیه ساز اندرویدی یا یک دیوایس حقیقی در حال اجرا (Run) است و می‌خواهیم تغییری در کدها اعمال کنیم، بهتر است بجای Run کردن دوباره‌ی پروژه از گزینه Apply Changes استفاده کنیم. با اجرای مجدد پروژه، اپلیکیشن به طور کامل ریستارت شده و به عبارتی یک APK جدید ساخته می‌شود که جایگزین APK ای که در اجرای قبلی روی دیوایس نصب شده خواهد شد. این یعنی اتلاف زمان و منابع.

در صورتی که گزینه Apply Changes صرفا بخشی از برنامه که کدهای آن ویرایش شده را به دیوایسی که پروژه روی آن در حال اجراست ارسال می‌کند و یک APK جدید جایگزین نخواهد شد.



قابلیت Apply Changes در اندروید استودیو ۳.۵ معرفی و جایگزین Instant Run شد که در نسخه ۲ اضافه شده بود. بر اساس مستندات توسعه دهندگان اندروید استودیو، برخلاف Instant Run که تغییرات انجام شده در پروژه را از طریق بازنویسی بایت کد APK روی دیوایس اعمال می‌کرد، در Apply Changes کلاس‌های موردنظر دوباره تعریف می‌شوند که فرایند بهینه‌تر و کوتاهتری حاصل می‌شود.

البته استفاده از این قابلیت صرفا بر روی دیوایس‌های با Android 8 (API 26) و به بالا قابل انجام است. ضمن اینکه اعمال برخی تغییرات در پروژه‌در در حال اجرا صرفا با ریستارت شدن یا به عبارتی Run کردن دوباره پروژه انجام خواهد شد و Apply Changes کاربردی نخواهد داشت مانند حذف یا اضافه کردن یک متد، فیلد یا کلاس.

توضیحات تکمیلی را در صفحه Build and Run مستندات اندروید مطالعه کنید.

انجام تغییرات در Gradle

با فعال کردن تعدادی از قابلیت‌های گریدل می‌توان باز هم افزایش سرعت بیلد Gradle در اندروید استودیو را شاهد باشیم. این کدها در فایل gradle.properties اضافه می‌شود:

Gradle Daemon: این قابلیت یک فرایند پس زمینه‌ای است و فعال کردن آن باعث می‌شود برای عملیات بیلد پروژه، گریدل حافظه بیشتری در اختیار داشته باشد که به کاهش زمان انجام عملیات منجر خواهد شد.

```
org.gradle.deamon=true
```

اگر مایلید در این زمینه اطلاعات کاملتری کسب کنید صفحه Gradle Daemon را مطالعه کنید.

Parallel Build Execution: به طور پیش فرض گریدل تنها یک کار را در آن واحد انجام می‌دهد اما با اضافه کردن خط زیر، گریدل چندین عملیات را همزمان و به طور موازی اجرا خواهد کرد.

```
org.gradle.parallel=true
```

البته این قابلیت تنها هنگامی موثر است که پروژه ماژولار ساخته شده باشد. آشنایی بیشتر با این قابلیت در این صفحه.

Configure On Demand: احتمال اینکه این قابلیت مورد استفاده ما قرار گیرد بسیار کم است زیرا صرفا در پروژه‌ای که اصطلاحا Multi-project نامیده می‌شود کاربرد خواهد داشت. پروژه‌ای که خودش دارای چند زیرشاخه مانند mobile و tv باشد.

به بیان ساده تر، هنگامی که بخواهیم این Multi-project را روی یک دستگاه موبایل اجرا کنیم، فقط همین قسمت بیلد شده و قسمت مربوط به tv بیلد نخواهد شد.

```
org.gradle.configureondemand=true
```

اطلاعات بیشتر در این زمینه: اینجا

تخصیص حافظه بیشتر به کامپایلر جاوا: می‌توانیم مقدار حافظه در دسترس برای کامپایلر را افزایش دهیم تا فرایند کامپایل با سرعت بالاتری انجام شود. برای مثال در خط زیر مقدار ۲ گیگابایت (۲۰۴۸ مگابایت) تعیین شده است.

```
org.gradle.jvmargs=-Xmx2048m -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -
Dfile.encoding=UTF-8
```

بر اساس مقدار حافظه سیستم می‌توان عدد بزرگتری نیز در نظر گرفت.

فعال کردن Gradle build cache: با فعال شدن قابلیت کش، گریدل از خروجی بیلدهای قبلی پروژه استفاده می‌کند که می‌تواند تا حدود ۳ برابر برای یک بیلد کامل (Full build) و ۱۰ برابر برای یک اعمال تغییر کوچک در پروژه (Incremental build) افزایش سرعت و کاهش زمان بیلد را به همراه داشته باشد.

```
org.gradle.caching=true
```

نکته: علاوه بر کش بیلد گریدل، خود اندروید نیز یک سیستم کش دارد که تا قبل از انتشار اندروید استودیو ۲.۳ توسط خط android.enableBuildCache=true فعال می‌شد. اما از این نسخه به بعد این قابلیت به صورت پیش فرض فعال بوده و نیازی به اضافه کردن آن نیست.

غیر فعال کردن PNG Crunching

هنگام بیلد شدن پروژه برای کاهش حجم فایل اپلیکیشن چنانچه تصاویری با فرمت PNG در منابع وجود داشته باشد فشرده خواهند شد که این فرایند به مدت زمان بیلد می‌افزاید. مسلما این قابلیت

برای نسخه release نهایی که قصد انتشار آن را داریم یک مزیت محسوب می‌شود اما در هنگام تست و عیب یابی پروژه طبیعتا کاهش حجم اپ اهمیتی برای ما نخواهد داشت. لذا با غیر فعال کردن آن تا حدودی از زمان بیلد کاسته خواهد شد.

البته از اندروید استودیو ۳ به بعد این قابلیت به طور پیش فرض برای خروجی debug غیر فعال بوده و صرفا برای نسخه release انجام این کار لازم است. در بلاک release داخل بلاک BuildTypes فایل build.gradle(app) برای ویژگی crunchPngs مقدار false تعریف می‌کنم. اگر به یاد داشته باشید تنظیمات مربوط به فعال کردن پروگارد (R8) نیز در همین بلاک قرار داشت.

```
buildTypes {
    release {
        crunchPngs false
    }
}
```

البته شاید بهتر باشد بجای استفاده از تصاویر PNG از همان ابتدا از فرمت جدید تصاویر با نام WebP استفاده کنیم تا اولا نیازی به غیر فعال کردن موقت crunchPngs و فعال کردن مجدد آن هنگام release نهایی نداشته باشیم و دوما در همه حالت‌ها چه debug و چه release تست و نهایی، حجم برنامه ما در کمترین حالت ممکن از نظر resource ها قرار بگیرد.

WebP فرمتی است که عمدتا در وب در حال فراگیر شدن بوده و جایگزینی آن با فرمت PNG به مرور در حال افزایش است. اگر با واژه سئو (SEO) آشنا هستید، یکی از آیتم‌های مهم در بهبود وضعیت سئو یک وب سایت، کاهش حجم صفحات و بخصوص تصاویر درون آن به شمار می‌رود.

با اینحال چنانچه به هر دلیلی استفاده از WebP برایتان مقدور نیست، غیر فعال کردن crunchPngs به عنوان راهکار جایگزین در دسترس قرار دارد.

جلوگیری از Legacy Multidex

قبل از پرداختن به این مورد خلاصه بگویم؛ اگر برای تست و دیباگ پروژه اندرویدی خود صرفا از دیوایس‌های مجازی یا حقیقی با API 21 (Android 5.0) و به بالا استفاده می‌کنید مشکلی وجود ندارد و بدون مطالعه این قسمت از آن عبور کنید.

قبلا در مبحث آشنایی با سیستم عامل اندروید گفتیم که از نسخه ۵.۰ اندروید، ران تایم ART جایگزین Dalvik شد. محدودیتی که در Dalvik وجود دارد این است که در یک فایل DEX (که بعد از کامپایل پروژه درون APK ساخته می‌شود) حداکثر 64k یا به عبارتی ۶۵۵۳۶ متد می‌تواند ارجاع داده شود.

لذا چنانچه در حال توسعه یک پروژه با مقیاس بزرگ هستیم و مجموع تعداد متدهای تعریف شده در پروژه (شامل کدها، کتابخانه‌ها و...) بیشتر از این عدد باشد و minSdkVersion هم روی API 20 و یا پایینتر تنظیم شده باشد، لازم است کتابخانه Multidex روی پروژه نصب و فعال شود تا متدها در دو یا چند فایل DEX قرار گیرد.

این آموزش جایی برای پرداختن به نحوه فعالسازی این کتابخانه نیست و چنانچه یکی از ارورهای

trouble writing output:

Too many field references: 131000; max is 65536.

You may try using --multi-dex option.

و یا

Conversion to Dalvik format failed:

Unable to execute dex: method ID not in [0, 0xffff]: 65536

گرفته‌اید صفحه فعالسازی Multidex را در مستندات اندروید مطالعه کنید.

بهتر است قبل از فعالسازی Multidex یکبار پروژه را بازنگری کنید. اگر کد یا کتابخانه اضافی هست که امکان حذف آن وجود دارد و با اینکار، تعداد متدها از محدودیت اعلام شده کمتر خواهد شد، نسبت به استفاده از Multidex در اولویت خواهد بود.

فعال بودن این قابلیت نه تنها باعث افزایش سرعت بیلد Gradle در اندروید استودیو نمی‌شود بلکه با کاهش سرعت و افزایش زمان بیلد مواجه خواهیم شد. لذا یا باید minSdkVersion را روی API 21 و به بالا تنظیم کرد و یا اینکه پروژه را روی دیوایس‌های Android 5.0 به بالا اجرا و تست کنیم تا Multidex دخالتی در فرایند بیلد شدن پروژه نداشته باشد. ضمن اینکه نباید از اندروید استودیو نسخه ۲.۳ و پایینتر از آن استفاده کرد.

جلوگیری از کامپایل منابع غیر ضروری

اگر برنامه‌ای که در حال توسعه آن هستید شامل چند زبان مانند فارسی و انگلیسی بوده و یا برای چندین اندازه صفحه طراحی شده، هنگام بیلد شدن پروژه تمامی این resource ها کامپایل می‌شود. در صورتی که هنگام تست و دیباگ ما فقط به یک زبان و یک اندازه تراکم صفحه نیاز داریم.

بنابراین با محدود کردن این موارد در هنگام تست و دیباگ پروژه می‌توانیم از اتلاف زمان برای کامپایل شدن منابع اضافی و غیر ضرور اجتناب کنیم.

یک بلاک با نام productFlavors در بلاک android در build.gradle(app) اضافه می‌کنم. تنظیماتی که مدنظر دارم فقط باید در هنگام تست پروژه اجرا شود و نه در زمان گرفتن خروجی APK یا AAB برای انتشار. بنابراین یک بلاک جدید با نام dev در productFlavors تعریف کرده و تنظیمات موردنظرم را داخل آن اضافه می‌کنم:

```
android {
    ...
    productFlavors {
        dev {
            resConfigs "fa", "xhdpi"
        }
    }
}
```

در مثال فوق هنگام Run شدن پروژه و یا گرفتن نسخه debug فقط زبان فارسی و اندازه صفحه (تراکم صفحه) xhdpi کامپایل شده و مابقی موارد چشم پوشی خواهد شد.

عدم استفاده از مقادیر داینامیک

اگر در مواردی مانند تعیین versionCode در پروژه اندرویدی خود از مقادیر داینامیک بجای استاتیک استفاده می‌کنید، در عین حال اینکه از انجام یک کار روتین راحت می‌شوید اما باید بدانید که به مدت زمان بیلد پروژه افزوده خواهد شد.

```
def buildTime = new Date().format('yyMMddHHmm').toInteger()

android {

    defaultConfig {

        versionCode buildTime

    }

}
```

در کد فوق از کلاس Date جاوا برای تعیین versionCode استفاده شده. بهتر است از انجام این کار اجتناب کرده و کد را به صورت دستی تغییر دهید.

غیر فعال کردن Crashlytics

اگر برای دریافت گزارشات کرش برنامه خود از ابزار Crashlytics استفاده می‌کنید اما نیازی نمی‌بینید هنگام دیباگ فعال باشد بهتر است این پلاگین در نسخه‌های debug غیر فعال شود تا از تاثیر منفی آن در مدت زمان بیلد شدن پروژه جلوگیری گردد.

برای اینکار خط زیر را در بلاک debug می‌کنم:

```
android {
    ...

    buildTypes {

        debug {

            ext.enableCrashlytics = false

        }

    }

}
```

اما اگر نیاز داریم این پلاگین در هنگام دیباگ هم فعال باشد باز هم با جلوگیری از بروز شدن Build id پلاگین Crashlytics در هر بیلد، می‌توان تا حدودی از زمان بیلد را کاهش داد:

```
android {
    ...

    buildTypes {

        debug {

            ext. alwaysUpdateBuildId = false

        }

    }

}
```

غیر فعال کردن Multi APK

اگر از قابلیت Multi APK برای ساخت چندین نسخه از اپلیکیشن برای دیوایس‌های با مشخصات سخت افزاری متفاوت استفاده می‌کنید، قطعا در هنگام تست و دیباگ نیازی به این قابلیت نیست و بهتر است غیر فعال شود.

کافیست در بلاک debug دو خط زیر تعریف شود تا برای همه سایزهای صفحه نمایش (density) و همچنین معماری‌های CPU (abi) فقط یک APK ساخته شود:

```
buildTypes {
    ...
    debug {
        splits.abi.enable = false
        splits.density.enable = false
    }

}
```

جلوگیری از کامپایل مداوم کتابخانه‌ها

طبیعتا در حین توسعه یک اپلیکیشن بارها و بارها نیاز به اصلاح و تغییر کدها و تست مجدد آن روی دیوایس اندرویدی داریم. اما کتابخانه‌هایی که در پروژه استفاده شده بدون تغییر هستند و واقعا نیازی نیست با هر بار بیلد شدن پروژه، کتابخانه‌های داخل پروژه نیز دوباره بیلد شده و درون فایل DEX در کنار دیگر کدهای کامپایل شده قرار گیرد.

با افزودن بلاک dexOptions به بلاک android در build.gradle(app) و تعیین مقدار true برای ویژگی preDexLibraries از این پس کتابخانه‌ها فقط یکبار کامپایل شده و در دفعات بعد فقط کدهای پروژه مجدد کامپایل خواهند شد.

```
android {
    ...
    dexOptions {
        preDexLibraries true
    }
}
```

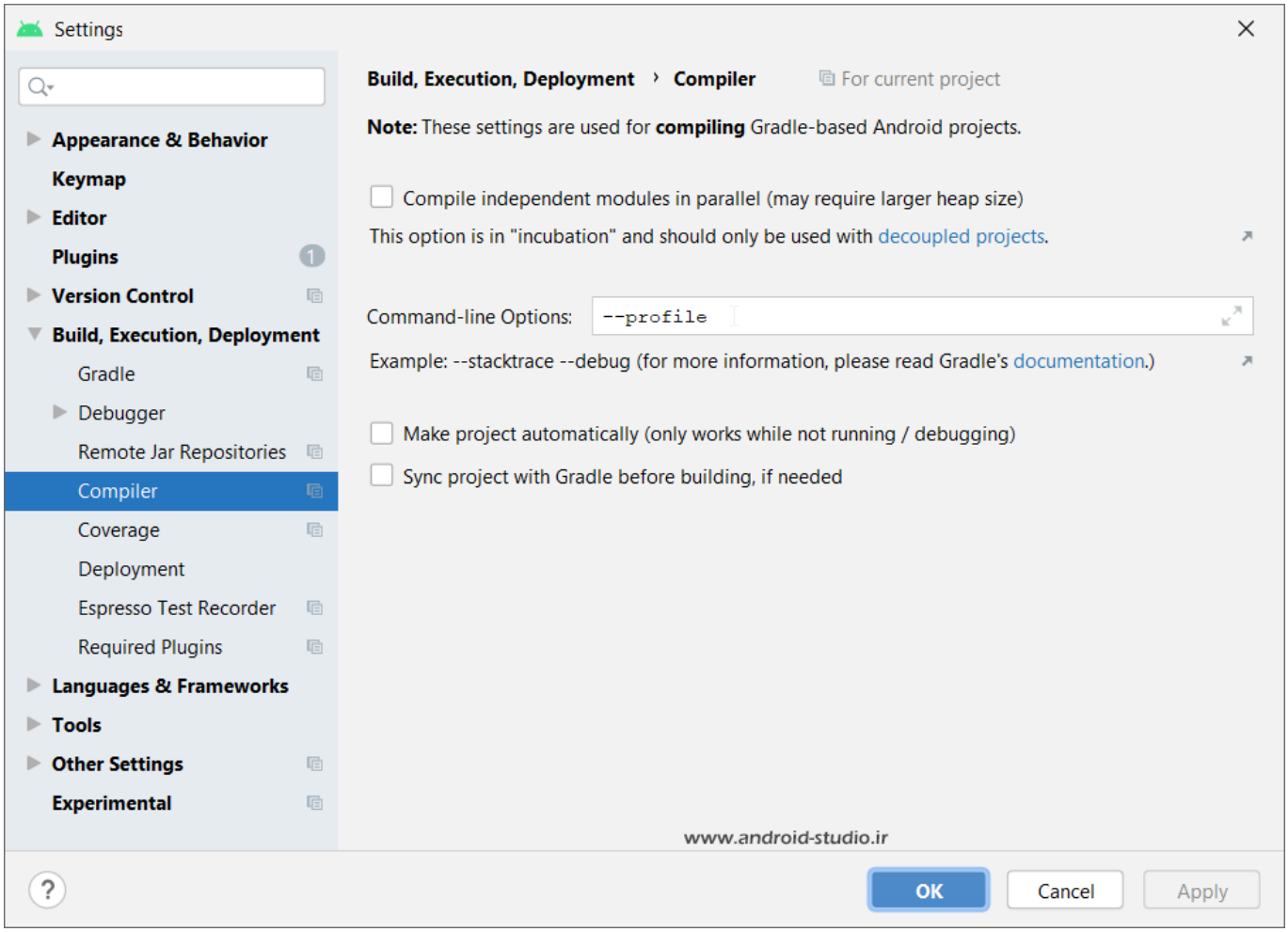
البته دقت داشته باشید این قابلیت فقط در تغییرات جزئی (Apply Changes) موجب افزایش سرعت بیلد Gradle در اندروید استودیو خواهد شد ولی در بیلدهای کامل می‌تواند باعث کاهش سرعت بیلد گریدل شود. بنابراین بهتر است هنگام بیلدهای کامل، این قابلیت غیر فعال (false) باشد.

ساخت پروفایل برای بیلد و بررسی آن

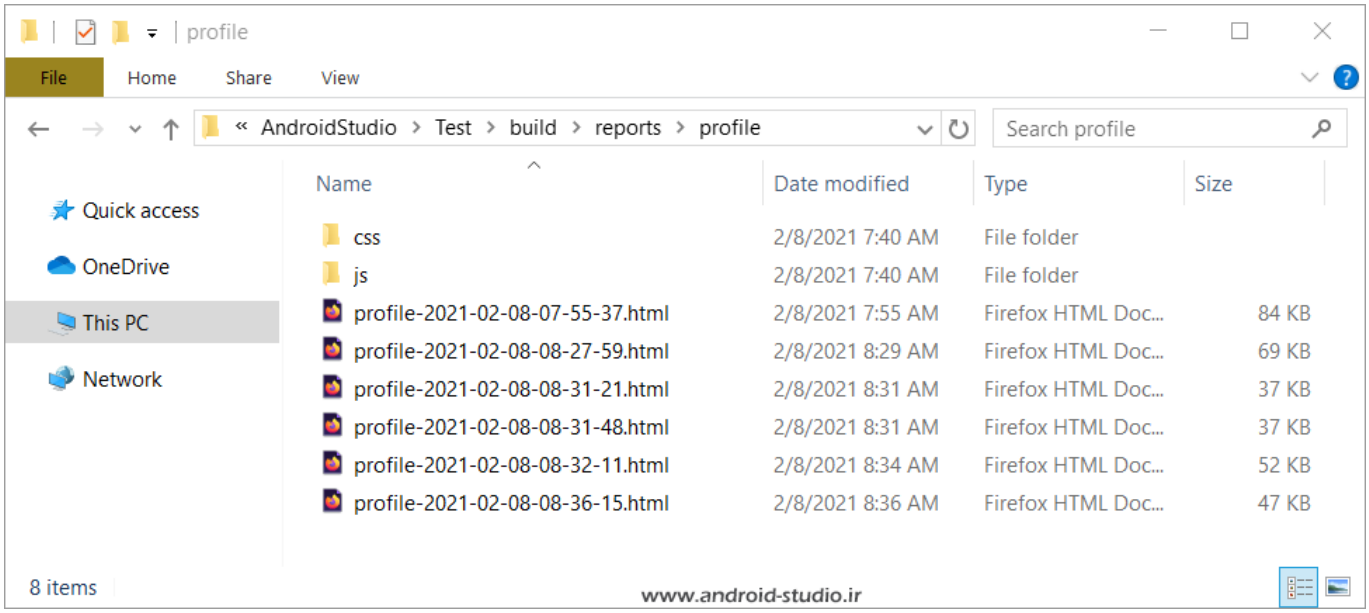
یکی از امکانات دیگری که Gradle در اختیار توسعه دهندگان قرار داده، ساخت Profile برای هر بیلد است. با فعال کردن پروفایل، بعد از هر خروجی APK ای که از پروژه می‌گیریم، یک گزارش جامع و کامل پیرامون مراحل بیلد و اینکه هر کدام چه مدت زمانی از فرایند کامل بیلد را به خود اختصاص داده در قالب یک فایل HTML در مسیر Project Directory > build > reports > profile ذخیره می‌شود.

برای فعال کردن profile دو راه وجود دارد:

۱. تعریف دستور profile-- در تنظیمات کامپایلر اندروید استودیو: کافیست یکبار دستور profile-- را در قسمت Command-line Options در مسیر Settings > Build, Execution, Deployment > Compiler تعریف کنیم:



حالا با هربار اجرای پروژه روی شبیه ساز یا دیوایس واقعی و یا گرفتن APK نسخه debug یا release یک پروفایل با پسوند html در پوشه profile ذخیره خواهد شد:



با باز کردن هر فایل در مرورگر، تمامی جزئیات بیلد در ۵ دسته بندی متفاوت نمایش داده می‌شود:

Profile report

Profiled build: :app:assembleDebug

Started on: 2021/02/08 - 08:36:15

Summary | Configuration | Dependency Resolution | Artifact Transforms | Task Execution

Description	Duration
Total Build Time	8.587s
Startup	0.033s
Settings and buildSrc	0.017s
Loading Projects	0.006s
Configuring Projects	0.166s
Artifact Transforms	0.871s
Task Execution	12.141s

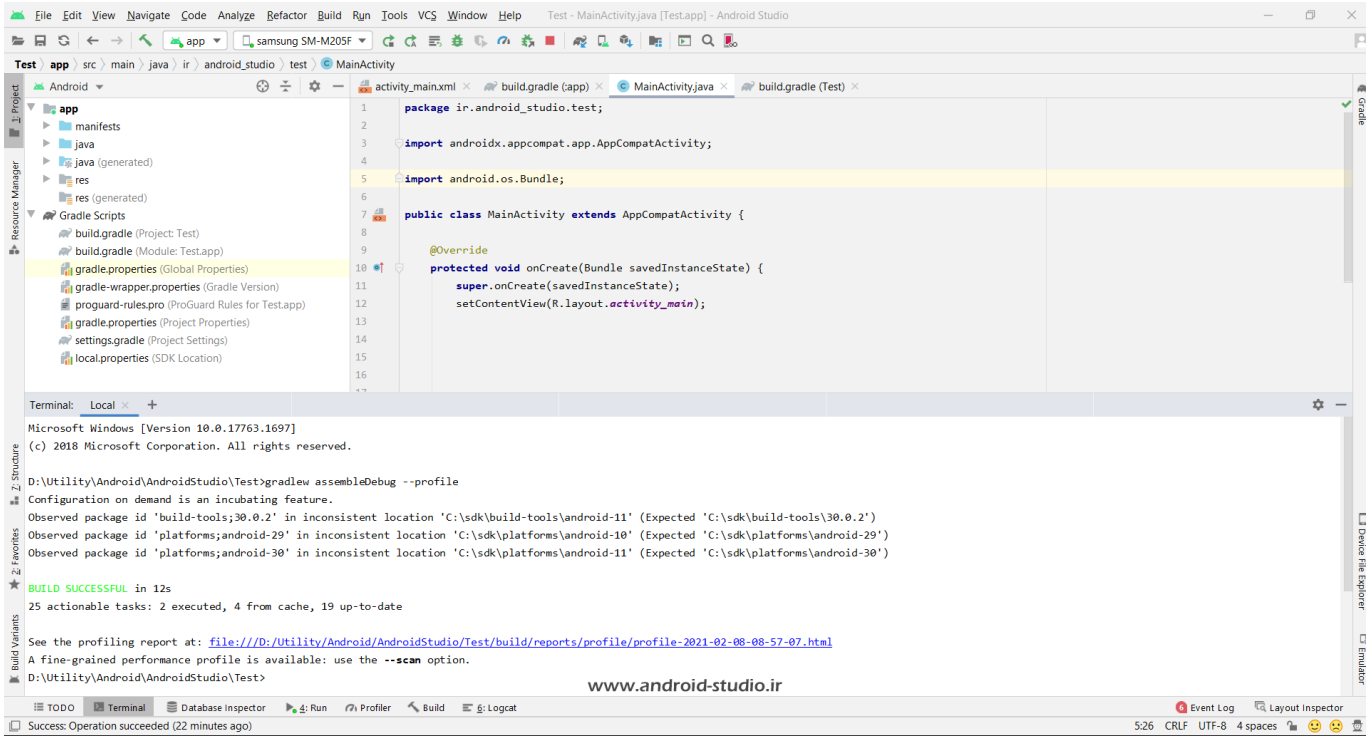
Generated by Gradle 6.5 at Feb 8, 2021 8:36:23 AM

www.android-studio.ir

در هر قسمت، عدد زیر Duration زمانِ مربوط به یک بخش از فرایند بیلد را نشان می‌دهد. با بررسی همه قسمت‌ها شامل Summary، Configuration، Dependency Resolution، Artifact Transforms و Task Execution می‌توانیم تشخیص دهیم چه مواردی بیشترین زمان بیلد را به خود اختصاص می‌دهد و کدامیک را می‌توان بهینه و یا کاملا حذف کرد.

۲. اضافه کردن profile-- به دستورات در خط فرمان: اگر عادت به استفاده از دستورات دارید، برای اضافه کردن پروفایل به بیلد باید profile-- به انتهای دستور اضافه شود:

```
gradlew assembleDebug --profile
```



ملاحظه می‌کنید پس از اجرای فرمان در Terminal اندروید استودیو، بیلد پروژه انجام شده و در انتهای آن لینک فایل html پروفایل هم نمایش داده شده است.

نکته: اگر تمایل دارید با ترمینال آشنا شوید و گزینه Terminal در نوار پایین اندروید استودیو فعال نبود، در منوی View > Tool Windows آنرا پیدا کنید. همچنین اگر بجای ترمینال اندروید استودیو از Command-line ویندوز استفاده می‌کنید، پس از بار کردن آن در پوشه پروژه، دستور را به این صورت وارد کنید:

```
./gradlew assembleDebug --profile
```

طراحی ماژولار پروژه

Modular که در فارسی ماژولار نامیده می‌شود بیانگر سبکی از طراحی و ساخت است که یک پروژه بزرگ از ترکیب چندین واحد کوچکتر تشکیل می‌شود که هر واحد را یک ماژول می‌نامیم.

تقسیم بندی برنامه به چند واحد (ماژول) علاوه بر مزایایی مانند توسعه و عیب یابی و بروزرسانی ساده تر و سریع تر و پیچیدگی کمتر، باعث می‌شود تا در فرایند تست و عیب یابی اپ، بیلد سیستم Gradle تنها ماژولی را کامپایل کند که ویرایش و اصلاح شده و مابقی ماژول‌هایی که ویرایش نشده‌اند برای

بیلدهای بعدی کش شود که می‌توان کاهش زمان بیلد گریدل را به مقدار زیادی در بیلدهای آتی شاهد بود.

با توجه به مفصل بودن طراحی ماژولار، به همین توضیحات مختصر اکتفا می‌کنم. اگر مایل هستید در مورد آن بیشتر مطالعه کنید از گوگل کمک بگیرید یا مقاله Modularizing Android Applications را مطالعه کنید.

در این آموزش ۲۱ عامل و روش افزایش سرعت و کاهش زمان بیلد Gradle در اندروید استودیو را بررسی کردیم. امیدوارم مورد مهم دیگری از قلم نیفتاده باشد. چنانچه یک یا چند روش دیگر را تجربه و یا در جایی دیگر مطالعه کرده‌اید، در قسمت دیدگاه‌های همین مبحث در وب سایت اطلاع دهید تا در بروزرسانی بعدی اضافه شود.

موفق و پیروز باشید.

مطالعه بیشتر:

https://androidstudio.googleblog.com/2017/06/android-studio-30-canary-5-is-now.html

https://developer.android.com/studio/build/optimize-your-build

با ارائه انتقادات و پیشنهادات خود، ما را در ارائه آموزش‌های بهتر یاری فرمائید.
این فایل رایگان بوده و انتشار آن (بدون دخل و تصرف) مانعی ندارد.

www.android-studio.ir