项目结构

1. 文件目录结构

1. gradle/.idea AS自动生成的一些文件
2. app
    1. build 编译时自动生成
    2. libs 项目中使用到的第三方jar包，该目录下所有jar包自动被添加到项目的构建路径里
    3. androidTest/test 编写AndroidTest测试用例/编写UnitTest测试用例
    4. main
        1. java 放置所有Java/Kotlin代码，主要的开发文件夹。负责整个app的逻辑处理，是完成整个app的核心所在，包括组件类、实体类、配置类、工具类、网络类所有开发代码内容都在此文件夹中。
        2. res 项目中使用到的所有图片、布局、字符串等资源，并以资源类别分小文件夹。存放非代码性质的资源文件，如图片、字体设置、UI布局、常量、网络配置、导航等。以文件用途、分辨率大小等因素划分为多个子文件夹，并以不同文件类型存放。图片多采用jpg/png/webp格式，其他文件则多用xml格式编写保存。
        3. AndroidManifest.xml 配置了Android程序的重要信息，名称、图标、包名、模块组成、权限、sdk版本等。
    5. build.gradle app模块的gradle构建脚本，指定项目构建相关的配置
    6. proguard-rules.pro 指定项目代码的混淆规则，当源代码打包为apk时通常会进行混淆代码，避免破解
3. build 编译时自动生成的文件
4. gradle 包含了gradle wrapper的配置文件，使用该方式可判断gradle的本地/网络使用
5. .gitingore git的忽略文件，指定目录/文件排除在版本控制之外
6. build.gradle 项目全局的gradle构建脚本
7. gradle.properties 全局的gradle配置文件，此处配置将影响项目所有gradle编译脚本
8. gradlew/gradle.bat gradle的命令行脚本
9. local.properties 指定本机中的Android SDK路径
10. settings.gradle 指定项目中所有引入的模块

2. 项目结构设置

File -> Project Structure

- Sdk Location 设置项目使用的JDK、SDK、NDK的位置
- Project 设置Gradle和Android Gradle Plugin的版本以及代码库位置名称
- Modules 修改特定于模块的构建配置，包括SDK版本、应用签名、库依赖项等

3. 项目主要构成

Android项目设计为逻辑、视图分离，因此在布局文件中编写界面，在组件中编写逻辑。

1. layout 位于/res/layout/文件夹中，主要是描述界面UI组件、边距、展示的xml文件，通过编写一系列原生或扩展的UI组件，调整布局关系，完成某一具体页面的布局、展示。
2. java 位于/modulename/java/文件夹中的内容，描述了app的功能逻辑、数据处理等，是app的功能核心文件夹，包括对activity、fragment、dialog等基础组件的功能开发等。

3. res　位于/module/java/res/文件夹中的内容，为整个app的资源型文件，如使用到的图片、出现的文字、动画等都分类存放于此，layout文件中仅通过引用的方式将这些资源文件导入进布局中。

配置

1. 各类配置文件

AS采用gradle构建项目，使用Groovy的领域特定语言（DSL）进行项目设置，抛弃了如IDEA使用Maven的XML的繁琐配置。

1. settings.gradle　位于根目录，用于指示Gradle构建时将哪些模块包含在内，多模块项目需要指定应包含在最终build中的每个模块。
2. 应用级build.gradle　位于根目录，定义适用于项目中所有模块的构建配置。默认情况下使用buildscript代码块定义共同的Gradle代码库、依赖项。

```
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:4.2.0'
    }
}
allprojects {
    repositories {
        google()
        jcenter()
    }
}
/*
添加额外的属性到顶层build.gradle的ext代码块中
*/
ext {
    sdkVersion = 28
    supportLibVersion = "28.0.0"
}
```

3. 模块级build.gradle

   位于每个project/module/目录下，为其所在的特定模块配置构建设置，可通过配置提供自定义打包选项。

```
apply plugin: 'kotlin-android-extensions'   //应用的插件
android {
    compileSdkVersion 31    //编译sdk版本
    ...
    defaultConfig { //默认配置：应用id、最小版本、目标版本...
        applicationId "com.example.demoapplicaiton"
        minSdkVersion 21
        ...
    }
    buildTypes {
        release {   //发布版本构建配置
            minifyEnabed false  //不开启代码混淆
            ...
```

```
        }
    }
}
dependencies {  //模块使用到的组件、库
    implementation fileTree(dir: "libs", include: ["*.jar"])
    ...
}
```

4. Gradle属性文件　位于根目录下，指定Gradle构建工具包本身设置
    - gradle.properties　配置项目全局Gradle设置

- local.properties　为构建系统配置本地环境属性

2. Gradle依赖项配置

```
1. implementation      只在内部使用此module，不向外部暴露其依赖的module内容
2. api      当前module会暴露其依赖的module内容，该方法依赖的库将参与编译、打包
3. compileOnly      该方法依赖的库仅在编译时有效，不参与打包
4. runtimeOnly      该方法依赖的库只在生成apk时参与打包，编译时不参与
5. testImplementation      仅在单元测试编译、apk打包测试时有效
6. debugImplementation      只在debug模式编译、debug apk打包时有效
7. releaseImplementation      仅针对release模式的编译、release apk打包有效。
```

> 依赖应首先设置为`implementation`，若不发生错误则不改变，若有错误再改变为`api`，这样会使编译速度增快。
>
> AS有3种依赖方式：本地依赖、库依赖、远程依赖。本地依赖可以对本地的jar包或目录添加依赖关系，库依赖可以对项目中的库模块添加依赖关系，远程以来可以对jcenter仓库上的开源项目添加依赖关系。
>
> gradle构建项目时会先检查本地是否有该库缓存，若没有则联网下载并添加到项目的构建路径中。

编译apk

构建流程

编译->转换（DEX）->打包->签名和对齐

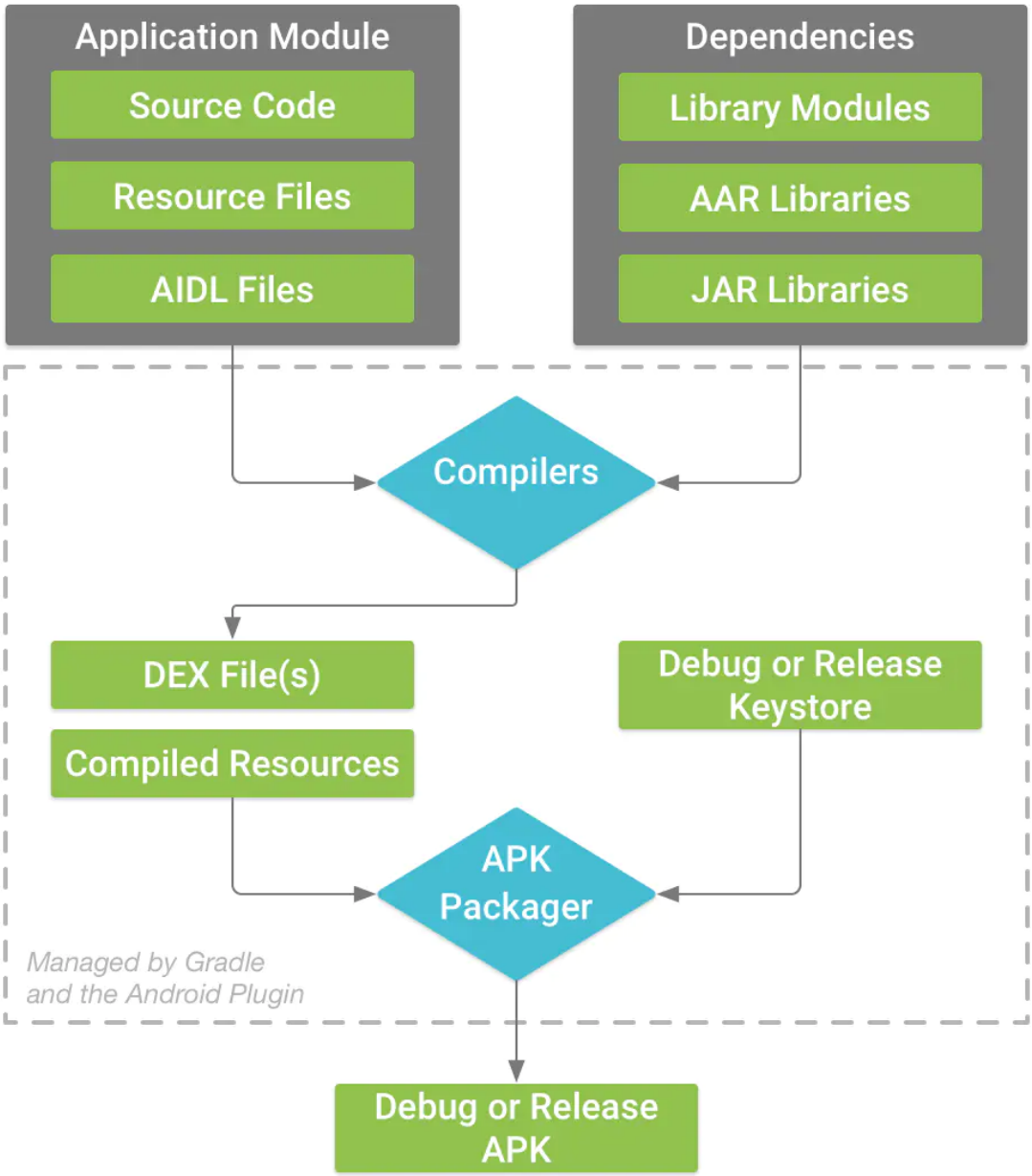
1. 编译、转换　将本工程的文件以及依赖的各种库文件进行编译，输出dex文件和编译后的资源文件

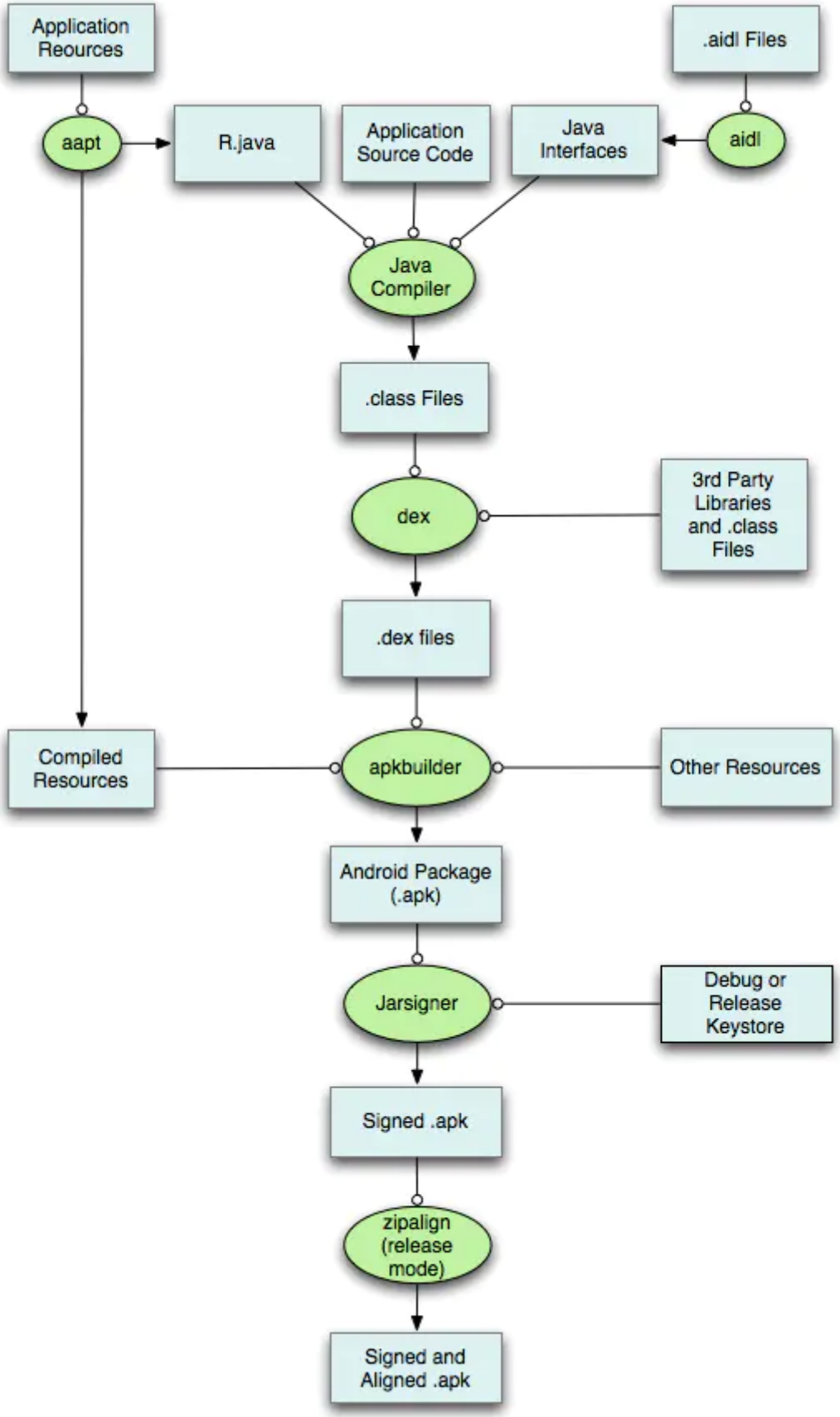
    1. Java编译器对工程本身的java、kotlin代码进行编译，有app源代码、资源文件生成的R文件(aapt工具)、java接口文件(aidl工具)，产出为.class文件
    2. .class文件和依赖的三方库通过dex工具生成Delvik虚拟机可执行文件，包含所有class信息
    3. 编译后：
        1. res资源目录编译后：
            - xml文件编译为二进制文件，提交解析速度
            - 图片被压缩
            - .9图的.9线会被去掉，.9线信息序列化保存进.9图中
        2. aapt -> 应用资源打包工具、生成R.java
        3. .java文件转换为.dex文件可缩减文件大小，classes.dex相关技术：64K方法数、APK瘦身、插件化、应用加固

2. 打包　打包器apkbuilder将DEX文件和编译后**资源整合**为APK或AAB文件。

> apkbuilder工具将.dex文件和编译后资源文件生成未经签名对齐的apk文件。编译后资源文件包括aapt产出资源文件和三方库资源文件。

3. 签名　打包器apkbuilder使用调试或发布密钥库keystore为APK、AAB签名：
    1. 调试版应用将使用调试密钥库签名，AS会自动使用配置新项目。
    2. 发布版应用将使用发布密钥库签名。
4. 对齐　生成最终APK前，打包器使用**zipalign**工具优化应用以减少内存占用。





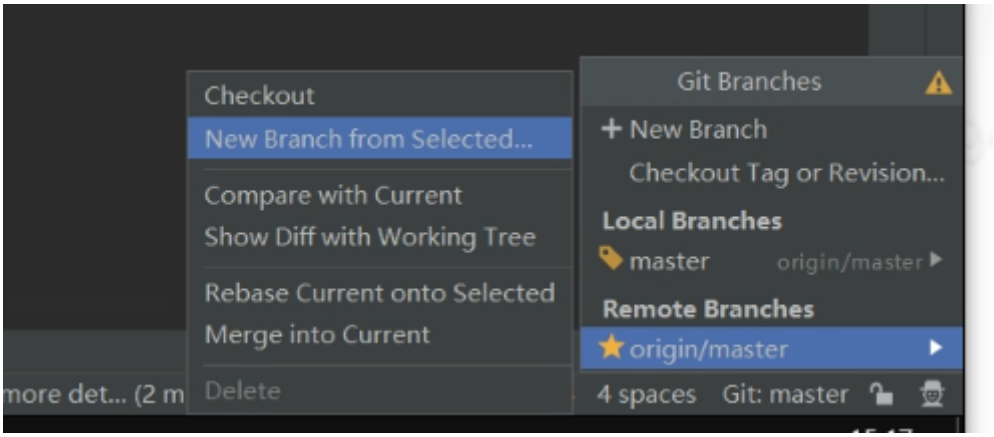
APK Analyzer

1. 简易的反编译工具
2. 可查看：应用的组成文件、内存占用大小、代码的总方法数、权限声明(通过查看AndroidManifest.xml文件)
3. 反编译后生成的文件：

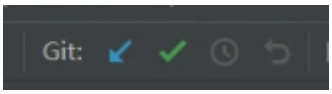
1. resources.arsc资源索引表　编译后的资源文件和索引，资源ID、资源名字、资源的值
2. META-INF/　存放校验文件

AS的可视化git

1. AS界面右下角自带可视化git插件，通过该插件可查看本地、远程分支，创建分支、跟踪分支、切换分支、比较差异、合并分支等操作。



2. AS界面右上角有git的四个按钮，可进行拉去最新版本、推送到远程分支、查看提交历史、回滚代码等操作。



Log/Logcat、Debug

1. Android使用Log作为日志工具类，重要程度由低至高分为v/d/i/w/e，分别对应verbose(最琐碎、意义最小)、debug(调试信息)、info(比较重要的数据)、warn(警告信息)、error(错误信息)。使用log而非println，日志开关可控制、可添加标签、有级别区分。Logcat可添加过滤器、控制日志级别、关键字过滤等，可快速筛选需要关心的日志。
2. 调试分类：
    1. 使用Debug模式启动程序　调试模式下，运行效率大大降低，需要额外操作，且操作感官上卡顿。
    2. 使用Attach Debugger to Android Process按钮开始调试　可以在手机上先运行到断点之前的流程，再开始调试，能快速、动态的到达断点位置。

使用技巧

1. 查找类技巧

1. 文件结构弹窗　Ctrl+F12 展示当前类的大纲，且可以快速跳转，迅速跳转到指定方法。
2. 书签功能　Ctrl+F11/Shift+F11　添加、删除书签/显示书签，显示警告或错误，快速定位错误。
3. 历史记录　Ctrl+Shift+E　显示最近本地修改过的文件列表
4. 查找　Ctrl+F　寻找某个关键字的位置
5. 替换　Ctrl+R　将所查找到的单词替换掉

2. 编码类技巧

1. 手动唤出代码提示框　Ctrl+Alt+Space
2. 快速生成父类方法　Ctrl+O　快速重写hashCode、equals等方法
3. 动态模板　Ctrl+J　快速插入代码片段的方法，可使用默认值将模板参数化