Android模块化开发

概念

在解决一个复杂问题时自顶向下逐层地把系统划分成若干模块的过程

简单来说就是将一个app分成不同的模块，不同的模块之间解耦，互不干扰，却又一起组成一个完整的app

组件化、插件化（可以自行了解）

优点

- 单独模块开发，编译迅速，调试方便
- 模块之间相互解耦，协同开发不容易出错
- 降低了编译时间，提高了开发效率

模块化开发的架构分层建议

命名建议：
底层：`library`
中间层：`module + 业务或功能名字`
上层：`App`

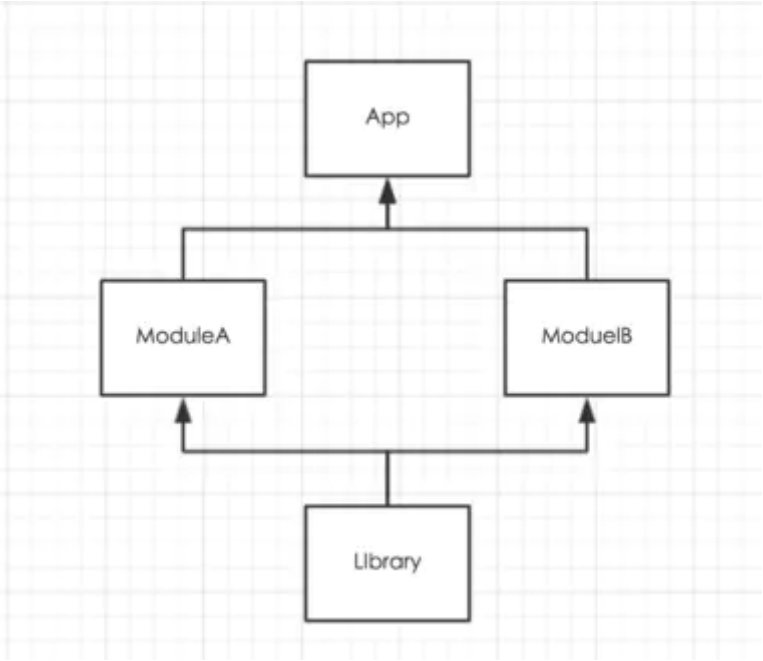
建议分层进行

- 底层：

  包含基础库和底层库

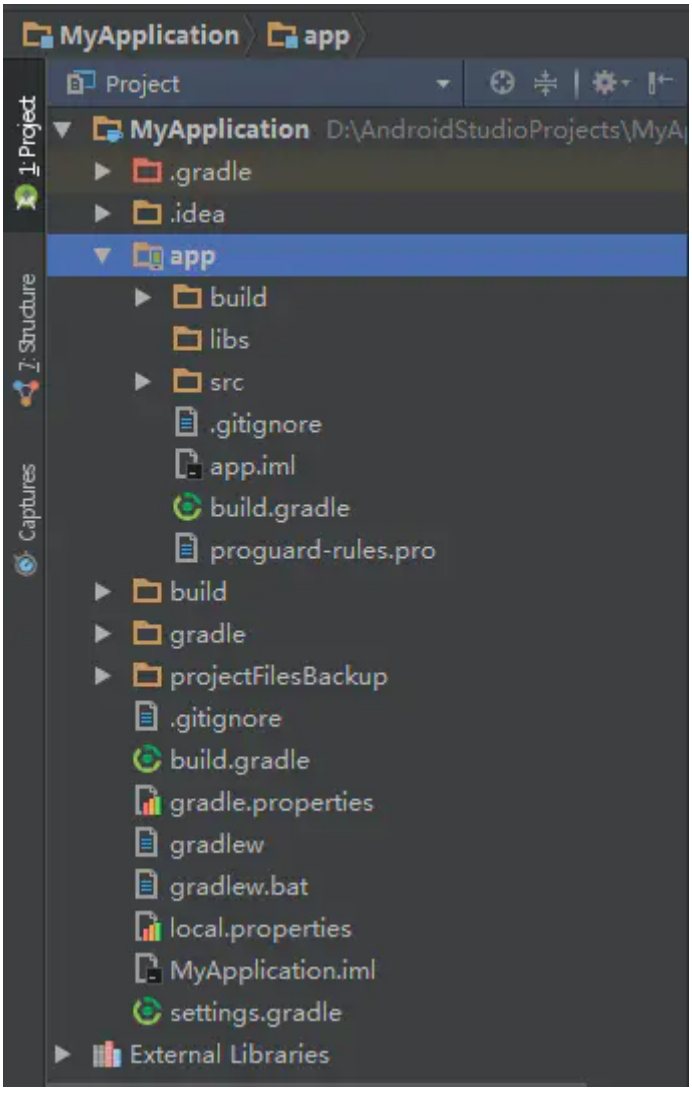
  - 基础库：
    包含所有模块需要的依赖库，以及一些工具类，比如封装了的常用网络请求，封装图片处理，数据库相关等，还包含所有模块需要的依赖库；
  - 底层库：
    主要是使用C/C++开发的跨平台的引擎或者库，以so的形式存在。例如：游戏引擎cocos2d。
- **中间层**
  首先、分模块肯定要按照功能分，独立的一个功能，不能杂。比如、更新、登录、分享、播放，都可以。

  之后，就可以按照一般写app的流程去编写界面，交互代码等
- **上层**
  将所有的业务模块聚合在一起，加上配置，形成主应用，一个模块化做的好的应用，主应用应该很简单，并且非常的稳定。



如何进行Android 模块化开发

在此之前，先复习一下Android Studio的项目结构

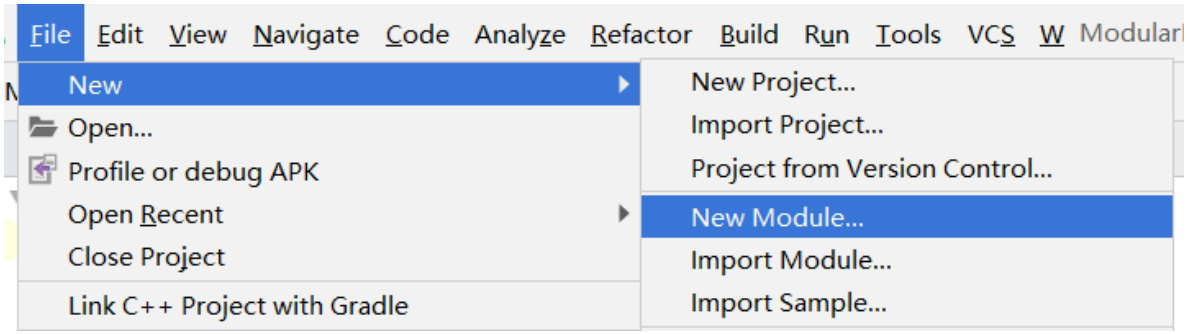


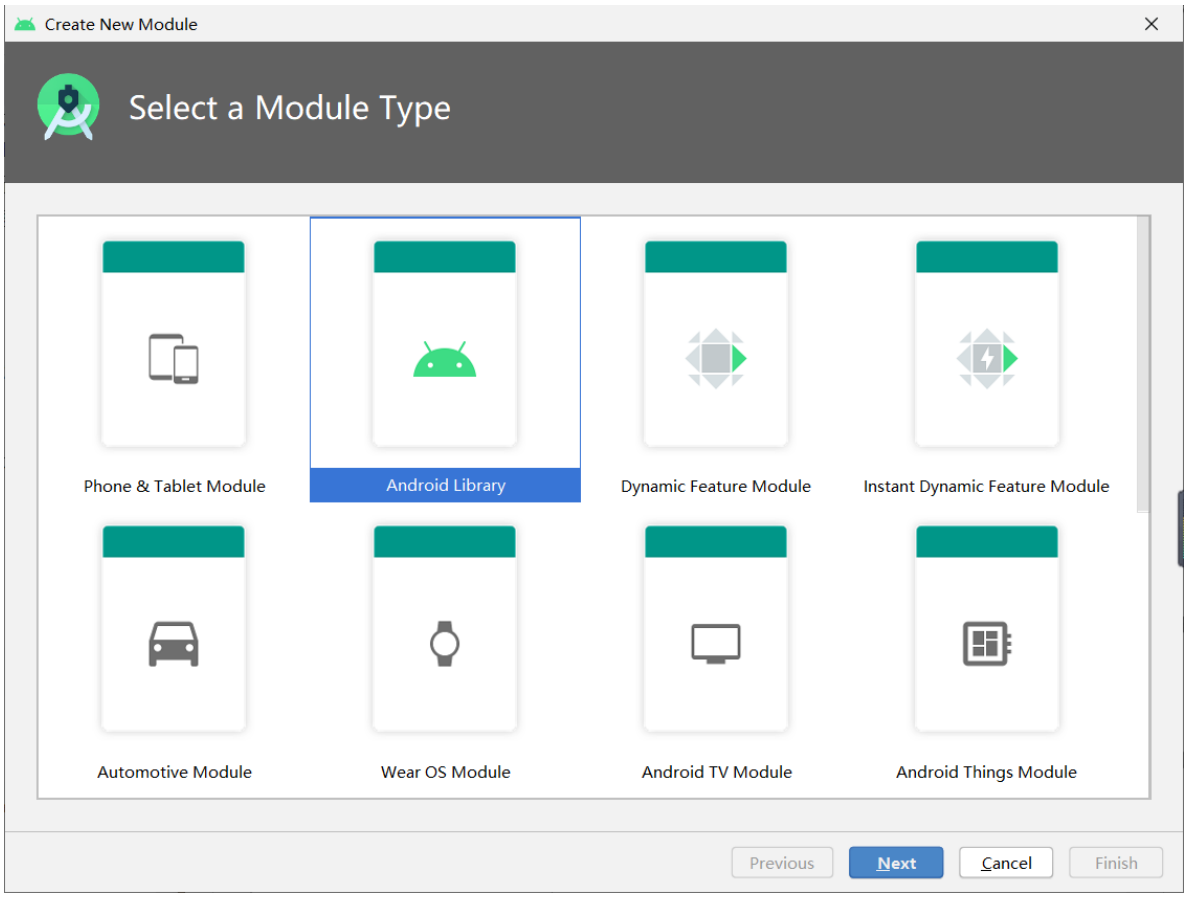
- MyApplication：整个项目目录
- app：项目中app模块目录
- settings.gradle：定义项目包含哪些模块

- app/build：app模块build编译输出的目录
- app.iml：app模块的配置文件
- External Libraries：项目依赖的Lib,编译时自动下载的

  可以看出 Studio 新建的项目自身就是一个模块化项目，MyApplication 是整个项目，而app是一个模块，所以在后续自行增加更多的功能模块。

创建Module模块





选择Android Library，点击Next，配置好名字等，然后finish就行了，很简单对吧？但实际上还有许多的地方需要注意.....

引用Module

创建好module之后，在其它的module里也是无法直接访问到该module的，需要引入，和引入依赖类似

```
dependencies {
    //其它的依赖

    implementation project(':module_name')
}
```

引入module、sync之后，就可以像第三方库一样使用module中的类了

模块间的跳转

上面介绍到app引用了其它的module，那它自然可以通过startActivity（）来跳转到需要的界面，这是可行的，但是其它的module呢？引用module，然后再startActivity（）跳转？

这显然不对，它违背了我们使用模块化开发的初衷。

子Module（非app）与父Module（app），子Module与子module之间的跳转有两种方式

1. 反射方式：通过反射可以调用集成在同一个APP中的另一个模块中的类或者方法，属性。因为反射是写死的类路径和类名，如果另一个模块中的类名变更的话，会导致反射找不到相对应的类，从而无法实现页面的跳转。
2. 使用路由框架：一般推荐阿里的ARouter

   ARouter的简单使用（以kotlin为例，kotlin与Java有差别）

   1. 添加依赖和配置

```
apply plugin: 'kotlin-kapt'//kotlin的特殊配置之一
android {
    //每个module都要配置
    defaultConfig {
        ...
        javaCompileOptions {
            annotationProcessorOptions {
                arguments = [AROUTER_MODULE_NAME: project.getName()]
            }
        }
    }
}
dependencies {
    // 替换成最新版本，需要注意的是api
    // 要与compiler匹配使用，均使用最新版可以保证兼容
    compile 'com.alibaba:arouter-api:x.x.x'//可只在base模块设置，引入到其它模块
    //在kotlin中annotationProcessor要更改为kapt
    annotationProcessor 'com.alibaba:arouter-compiler:x.x.x'
    ...
}
```

   2. 添加注解

```
// 在支持路由的页面上添加注解(必选)
// 这里的路径需要注意的是至少需要有两级，/xx/xx
@Route(path = "/test/activity")
class YourActivity :Activity() {
    ...
}
```

   3. 初始化SDK

```
if (isDebug()) {           // 这两行必须写在init之前，否则这些配置在init过程中将无效
    ARouter.openLog();     // 打印日志
    ARouter.openDebug();   // 开启调试模式(如果在InstantRun模式下运行，必须开启调试模式! 线上版本需要关闭,否则有安全风险)
}
ARouter.init(mApplication); // 尽可能早，推荐在Application中初始化
```

4. 跳转操作

```
//1.不含参数
ARouter.getInstance().build("/test/activity").navigation()

// 2. 含有参数
ARouter.getInstance().build("/test/1")
            .withLong("key1", 666L)
            .withString("key3", "888")
            .withObject("key4", new Test("Jack", "Rose"))
            .navigation()
```

5. 读取参数

```
val text = intent.extras?.getString("key3")
//其它方法请查看官方文档
```

其它的更多高级操作请自行查阅官方文档

模块间的通信

1. 广播
2. EventBus

模块间服务的调用

和activity跳转类似

- 添加注解

```
@Route(path = "/test/test")
  //这样ARouter框架才能找到这个服务
```

- 启动服务

```
ARouter.getInstance().navigation(ModuleService.class)
```

模块中 application 和 library 状态切换配置

在模块的build.gradle文件开始，有这样一段代码

```
apply plugin: 'com.android.library'
```

将library改成application就能实现application和library状态的切换了

用处：module是无法run的，只有转换成application，再配置好launch activity才能run，这样便能直接启动该界面进行调试、debug之类的，可以方便许多。

其它一些常见问题

资源名冲突：

1. 保护某些 resources 不被外部访问，可以创建`res/values/public.xml`，因为 public 是关键词，所以需要用 new file 的方式创建。至少添加一行，未添加的视为 private。
2. 在 library 的 build.gradle 中添加`resourcePrefix`，则所有的资源须以此 prefix 开头，否则报错。注意，图片资源虽然不提示报错误，但是也需要修改名字。

```
android {
    ...
    buildTypes {
    ...
    }
    resourcePrefix 'my_prefix_'
}
```

依赖重复： 将所有的依赖都写在library层的module，将所有的依赖同意成一个入口给上层的app去引用。