Android架构方案及ReactNative集成

一、插件化和组件化如何选择

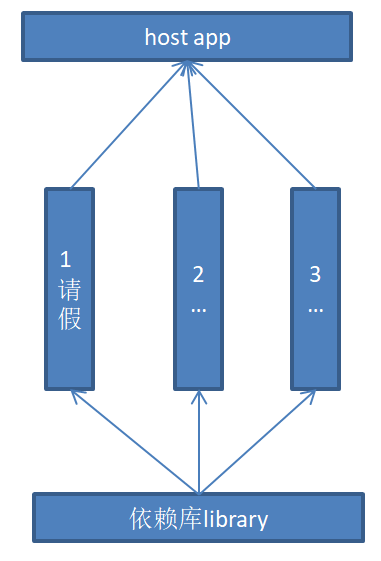
项目发展到一定阶段，随着人员的增多，代码越来越臃肿，这时候必须进行模块化的拆分。由于项目的庞大，耦合度高，拆分也是一个艰难的工作，所以在项目初始化阶段就要做好模块划分。具体到android领域，模块化开发是一个不错的选择，具体思想利用模块化将代码分而治之，解耦分层。具体实施分为插件化和组件化。针对组件化和插件化如何选择呢？插件化和组件化取舍的一个重要原则是：App是否有动态增加或修改线上模块的需求，如果这种动态性的需求很弱就不需要考虑插件化。

二、彻底组件化

彻底组件化主要有两点：针对接口编程、组件可以动态加载和卸载。只有实现了以上两点项目才可以彻底组件化。如何实现呢？定义好代码边界，避免直接组件之间的直接引用，(compile project)要坚决避免。根据彻底组件化的思想github已有开源的解决方案，在此方案下个人进行适配As3.1.2 https://github.com/zhaobingqing/jimu-sample-project，后续如进行彻底组件化可考虑采用。

三、本项目架构方案

脱离开项目谈架构都是不切实际的，由于本项目主页分为了6类模块，并且请假模块采用了reactNative进行实现，为了进行快速开发，采用(compile project)模式进行开发，将请假功能抽离成一个module，集成进host(宿主)app。后续如果新增模块的话建议采用彻底组件化方案。项目架构图如下：



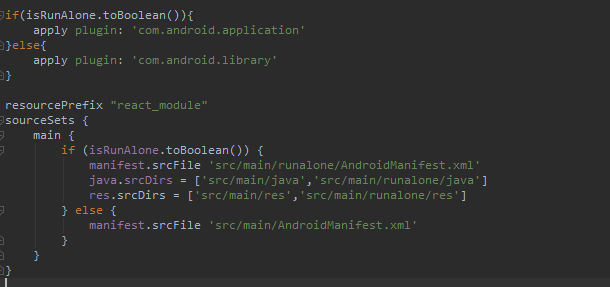
各模块说明：

1、下面为依赖库也作为一个module,主要集成第三方库及一些工具。

2、中间为各业务模块，通过配置可以作为host app的依赖，也可以单独运行。

3、host app为主模块，最终的输出apk。

四、配置依赖包单独运行，脱离对host app的依赖



单独调试需要修改一下AndroidManifest文件，因为单独调试需要一个入口activity。可以设置一个变量isRunAlone,标记当前是否需要单独调试，根据isRunAlone取值，使用不同的插件gradle插件和AndroidManifest文件，为了避免不同组件之间的资源名重复，在每个组件的build.gradle中增加resourcePrefix，固定每个资源的前缀。

五、集成

1、生成jsbundle

|  |
| --- |
| react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/com/your-company-name/app-package-name/src/main/assets/index.android.bundle --assets-dest android/com/your-company-name/app-package-name/src/main/res/ |
| 会生成对应的bundle文件和资源文件，拷贝出来放在对应的目录下 |

2、添加依赖module中

|  |
| --- |
| dependencies {  compile "com.facebook.react:react-native:+"  } |

3、根build.gradle

|  |
| --- |
| allprojects {  repositories {  mavenLocal()  jcenter()  maven {  // All of React Native (JS, Obj-C sources, Android binaries) is installed from npm  url "$rootDir/../node\_modules/react-native/android"  }  google()  }  } |

4、创建MyReactActivity

|  |
| --- |
| public class MyReactActivity extends Activity implements DefaultHardwareBackBtnHandler {  private ReactRootView mReactRootView;  private ReactInstanceManager mReactInstanceManager;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  mReactRootView = new ReactRootView(this);  mReactInstanceManager = ReactInstanceManager.builder()  .setApplication(getApplication())  //注意这里为生成的bundle的文件名可以修改  .setBundleAssetName("index.android.bundle")  .setJSMainModulePath("index")  .addPackage(new MainReactPackage())  .setUseDeveloperSupport(BuildConfig.DEBUG)  .setInitialLifecycleState(LifecycleState.RESUMED)  .build();  // 注意这里的MyReactNativeApp必须对应“index.js”中的  // “AppRegistry.registerComponent()”的第一个参数  mReactRootView.startReactApplication(mReactInstanceManager, "MyReactNativeApp", null);  setContentView(mReactRootView);  }  @Override  public void invokeDefaultOnBackPressed() {  super.onBackPressed();  }  } |

5、将activity的生命周期回传给ReactInstanceManager

|  |
| --- |
| @Override  protected void onPause() {  super.onPause();  if (mReactInstanceManager != null) {  mReactInstanceManager.onHostPause(this);  }  }  @Override  protected void onResume() {  super.onResume();  if (mReactInstanceManager != null) {  mReactInstanceManager.onHostResume(this, this);  }  }  @Override  protected void onDestroy() {  super.onDestroy();  if (mReactInstanceManager != null) {  mReactInstanceManager.onHostDestroy(this);  }  if (mReactRootView != null) {  mReactRootView.unmountReactApplication();  }  } |

6、传递后退事件给ReactNative

|  |
| --- |
| @Override  public void onBackPressed() {  if (mReactInstanceManager != null) {  mReactInstanceManager.onBackPressed();  } else {  super.onBackPressed();  }  } |

至此原生应用集成完reactNative!