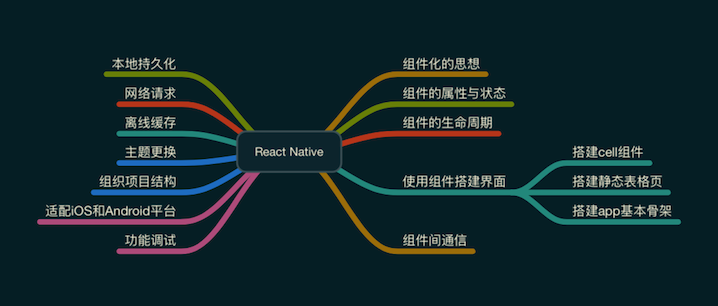
1. RN介绍
2. 环境配置
3. 基础知识
4. 生命周期
5. Props和State
6. Style样式和flex布局
7. 组件：flatlist Navigation
8. 触摸事件
9. 动画
10. 网络框架Fetch
11. 集成到现有应用
12. 使用原生模块和UI组件
13. 注意事项



1. RN介绍
2. 环境配置

（一）开发平台Windows 目标平台Android

1. 安装Chocolatry（Windows包管理器）

安装

@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-object net.webclient).DownloadString('https://chocolatey.org/install.ps1'))" && SET PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin

升级

choco upgrade chocolatey

2.安装Python 2 (不支持 python 3)

choco install python2

3.安装node

choco install nodejs.install

设置npm镜像

npm config set registry https://registry.npm.taobao.org --global

npm config set disturl https://npm.taobao.org/dist --global

4.Yarn、React Native的命令行工具（react-native-cli）

npm install -g yarn react-native-cli

设置yarn镜像

yarn config set registry https://registry.npm.taobao.org --global

yarn config set disturl https://npm.taobao.org/dist --global

yarn可以代替npm

5.安装AS

SDK Platforms: Android 8.1 (Oreo)

Android SDK Platform 27

Intel x86 Atom\_64 System Image

SDK Tools:

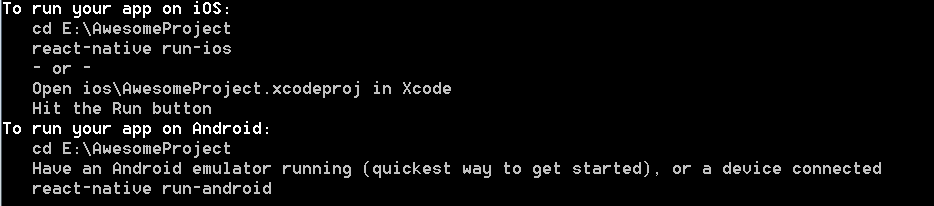
Android SDK Build-Tools 27.0.3（必须是这个版本）

6.ANDROID\_HOME环境变量

新建 ANDROID\_HOME 环境变量指向了Android SDK的路径。

（二）测试环境配置是否成功

1.react-native init AwesomeProject



2.cd AwesomeProject

3.设置端口react-native start --port 8080 --reset-cache

由于react-native默认使用8081端口 被占用



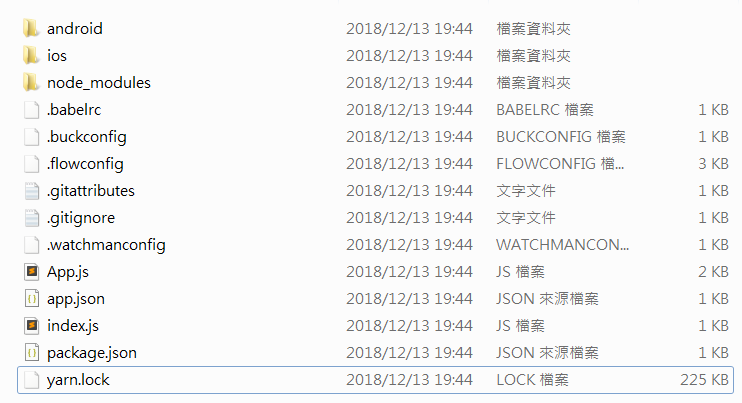
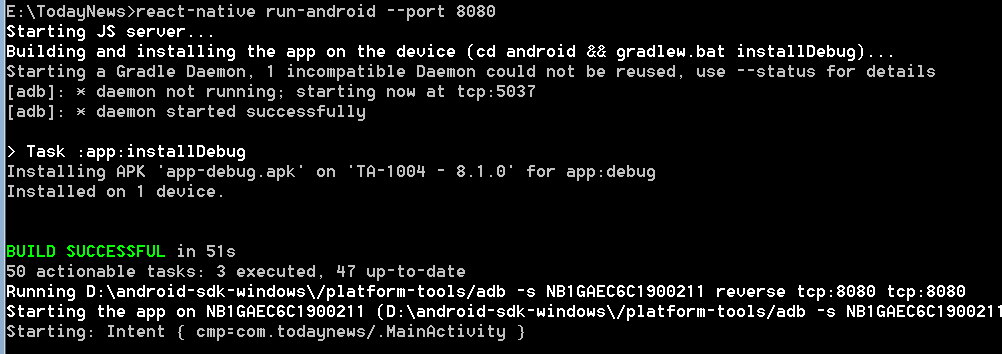
4.生成bundle文件

（1）在Android/app/src/main 建立文件夹assets

（2）返回工程根目录

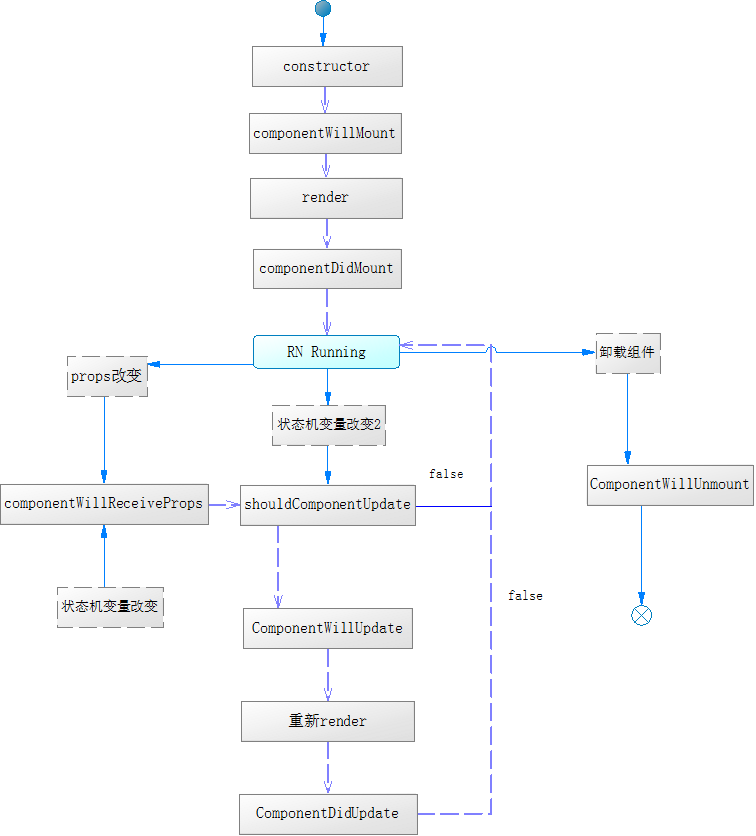
运行 react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index.android.bundle --assets-dest android/app/src/main/res 生成index.android.bundle文件

5.react-native run-android –port 8080



三、基础知识

（一）生命周期



1.初始化

（1）constructor（构造函数）

①第一个语句必须是super(props)。

②contructor将在任意一个RN组件被加载之前优先调用，并且只会调用一次。

③该函数最大的作用是定义该组件当中需要使用的状态机变量State。

（2）componentWillMount

①该函数原型为componentWillMount()

②该函数整个过程中只执行一次

③该函数会在初始渲染前执行，即在render被调用之前调用

④该函数执行后render就会调用。

⑤如果在render中改变了某些状态机变量State，那么RN不会执行渲染，而是等待该函数执行完毕后再次渲染。

⑥子组件中同样拥有该方法，并会在父组件执行完毕后执行，该函数无返回值。

⑦该函数适合于需要在本地读取一些数据用于显示，那么在render执行前调用是一个很好的时机。

（3）render(渲染函数)

①调用该方法，先对状态机变量State与属性Props进行检查。

②如果开发者不想渲染界面的话，可以在此处返回null或者false。

③该方法适用于进行界面的JSX代码编写，只允许返回一个最外层容器组件。

④尽量保持纯净，只渲染组件，不修改状态，不执行副操作（比如计时器）。

（4）componentDidMount

①该函数会在render渲染完毕之后调用，整个过程只执行一次。

②该函数执行后，开发者就可以对界面上的组件或者子组件进行各种操作了。

③该函数的应用场景适用于在移动端应用启动之后需要访问网络进行某些数据获取。

2. 运行中

（1）componentWillReceiveProps

①该函数的原型为componentWillReceiveProps(nextProps);

②当props(属性)发生改变或者接受到新的props时，该函数被调用，并接受一个输入参数，类型为Object，存放新的props，原先反生改变的旧的props仍然可以通过this.props访问。

③该函数在RN初次被渲染的时候不会被调用。

④如果在该函数当中对状态机变量进行了修改，RN不会立即渲染页面，而是会等待该方法执行完毕后一起渲染。

（2） shouldComponentUpdate:（组件是否需要更新）

①该函数原型 : boolean shouldComponentUpdate(nextProps,nextState);

②当props(属性)或者状态(state)发生改变的时候会触发该函数，分别对应接收的两个参数，根据返回的布尔值来决定是否需要对页面进行重新渲染，如果不进行渲染，那么该方法后续的componentWillUpdate与componentDidUpdate都不会被执行。

③该函数默认会返回true。

④应用场景：可以在该函数中编写一些逻辑来判断渲染类型，来阻值一些没有必要的重新渲染，达到提升应用运行效率的目的。

（3） componentWillUpdate

①该方法原型与上一方法相同，在重新渲染前会调用该方法，为渲染进行准备工作。

②该函数无返回值。

③注意：在该方法中，不应该对状态机变量进行修改，要改变，也应该在componentWillReceiveProps当中进行。

（4）render：跟初始化的时候功能一样。

（5）componentDidUpdate

该函数会在重新渲染render之后调用，传递过来的是当前的props和state。

3.销毁

componentWillUnmount

该方法会在RN卸载之前调用，无参无返回值，在该方法中，需要对该组件当中申请或者订阅的某些资源与消息进行释放。

（二）State和Props

1. State

（1）作用

　　State是RN中组件的一个对象，RN把用户界面当做是状态机，想象它有不同的状态然后渲染这些状态，可以轻松让用户界面与数据保持一致。

RN中，更新组件的state会导致重新渲染用户界面(不操作DOM)。简单来说，就是用户界面会随着state变化而变化。

（2）原理

常用的通知RN数据变化的方法是调用setState(data,callback)，这个方法会合并data到this.state，并重新渲染组件，渲染完成后，调用可选的callback回调。大部分情况不需要callback。

2. Props

组件中的props是一种父级向子级传递数据的方式。

3.联系

（1）相同点

①不管是props还是state的改变，都会引发render的重新渲染。

②都能由自身组件的相应初始化函数设定初始值。

（2）不同点

①初始值来源：state的初始值来自于自身的constructor函数；props来自于父组件或者自身getDefaultProps（若key相同前者可覆盖后者）。

②修改方式：state只能在自身组件中setState，不能由父组件修改；props只能由父组件修改，不能在自身组件修改。

③对子组件：props是一个父组件传递给子组件的数据流，这个数据流可以一直传递到子孙组件；state代表的是一个组件内部自身的状态，只能在自身组件中存在。



（三）Style样式和FlexBox布局

1. Style样式

使用JS来写样式，所有的核心组件都接受名为style的属性。这些样式名基本上是遵循了 web 上的 CSS 的命名，只是按照 JS 的语法要求使用了驼峰命名法，例如将background-color改为backgroundColor。

1. 声明

**import React, { StyleSheet } from "react-native";**

**const styles = StyleSheet.create({**

**base: {**

**width: 38,**

**height: 38,**

**},**

**background: {**

**backgroundColor: '#222222',**

**},**

**active: {**

**borderWidth: 2,**

**borderColor: ‘#ff00ff',**

**},**

**});**

1. 使用

①使用属性类

**<View style={ { width: 50 , height: 50 } } > </View>**

②单个属性类

**<View style={ styles.base } > </View>**

③多个属性类

**<View style={ [ styles.base , styles.backgroundColor ] } > </View>**

④混合使用（不推荐）

**<View**

**style={ [ styles.base, { width:this.state.width, height:this.state.width\*this.state.aspectRatio } ] }**

**/>**

1. 特点

①当属性设置冲突时，后边设置的属性会覆盖先前设置的属性。

②样式继承，布局嵌套时，里层的组件样式继承外层的。

**<View style={ styles.base } >**

**<View style={ styles. backgroundColor } >**

**</View>**

**</View>**

1. 属性



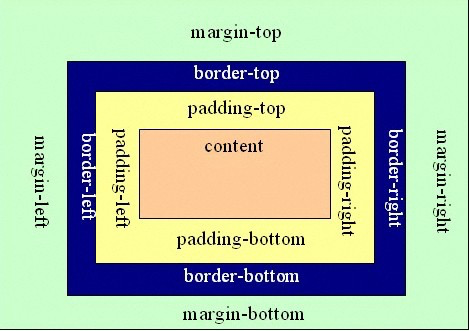


2. FlexBox布局

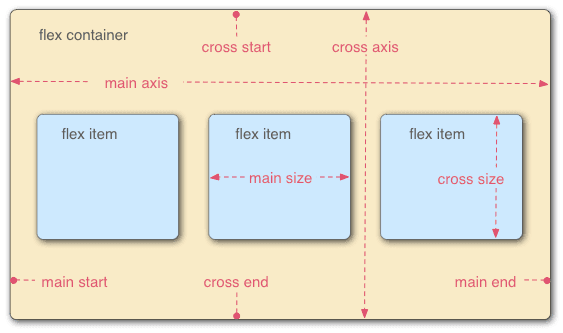
（1）FlexBox介绍

Flexbox是Flexible Box的缩写，意为'弹性布局'，它为盒状模型提供了很大的灵活性，让任何一个容器都可以指定为Flex布局。

使用Flex布局的元素，称为Flex容器（flex Container），简称'容器'。它的所有子元素自动成为容器的成员，称为Flex项目（flex Item）。



标准的盒子模型



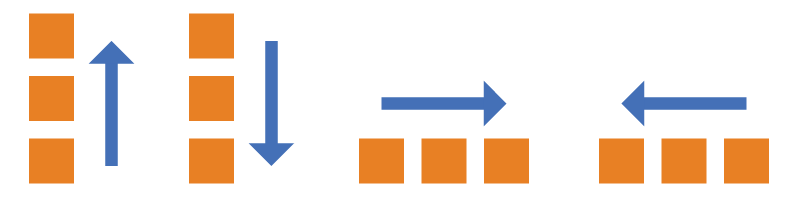
FlexBox布局的示例图

Flex容器默认存在两根轴：水平方向的主轴(main axis)，与主轴垂直方向的交叉轴(cross axis)。主轴的排布方向是从左至右，交叉轴的排布方向是从上至下，Flex容器中的Item元素默认是按照主轴方向进行排序，Item占据主轴空间称为main size，Item占据交叉轴空间称为cross size。

（2）Flex属性

①flexDirection用来决定容器的主轴方向

* column-reverse：主轴为竖直方向，起点在下沿
* column（默认）：主轴为竖直方向，起点在上沿
* row：主轴为水平方向，起点在左端
* row-reverse：主轴为水平方向，起点在右端

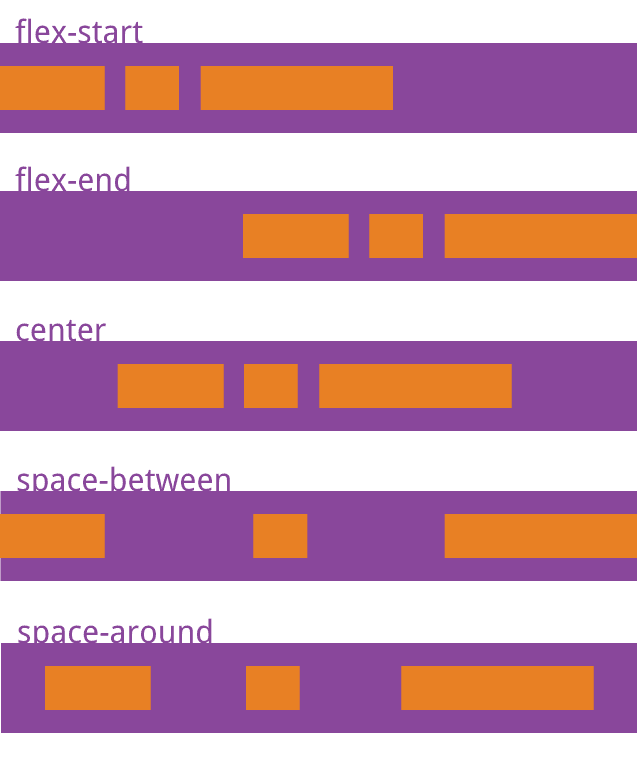


②flexWrap 进行换行排列

* nowrap（默认值）：不换行
* wrap：换行，第一行在上方
* wrap-reverse：换行，第一行在下方

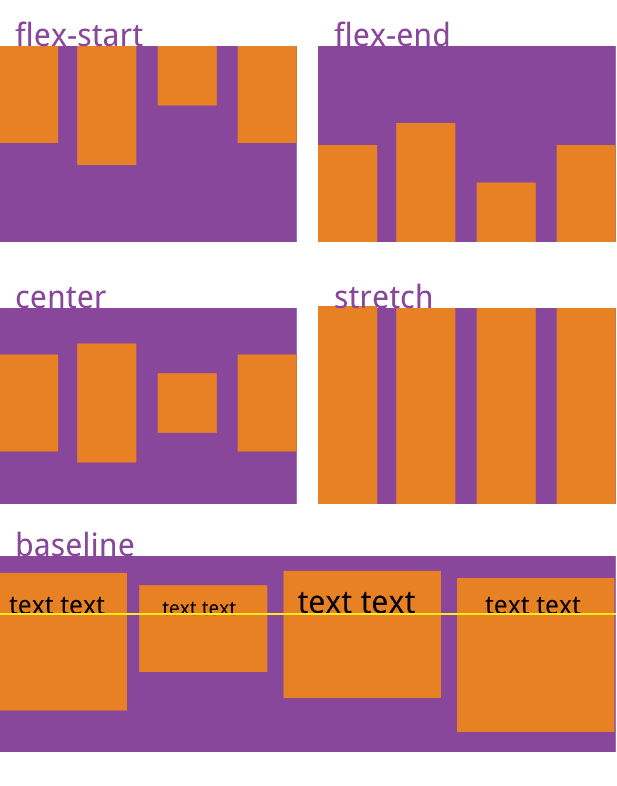
③justifyContent Item在主轴方向上的对齐方式

* flex-start（默认）：左对齐
* flex-end ：右对齐
* center：居中
* space-between：两端对齐，Item之间的间隔都相等
* space-around：每个Item两侧的间隔相等。Item之间的间隔比Item与边框的间隔大一倍



④alignItems Item在交叉轴上的对齐方式

* flex-start：（默认值）交叉轴的起点对齐
* flex-end：交叉轴的终点对齐
* center：交叉轴的中点对齐
* baseline: 项目的第一行文字的基线对齐
* stretch：如果项目未设置高度或设为auto，将占满整个容器的高度



⑤alignself Item自身对齐

除了多一个auto属性，其余与alignItems属性相同

（四）组件

React认为一个组件应该具有如下特征：

可组合（Composeable）：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有它创建的子组件，通过这个特性，一个复杂的UI可以拆分成多个简单的UI组件；

可重用（Reusable）：每个组件都是具有独立功能的，它可以被使用在多个UI场景；

可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；

1. FlatList

常用功能：

* 完全跨平台
* 支持水平布局模式
* 行组件显示或隐藏时可配置回调事件
* 支持单独的头部组件
* 支持单独的尾部组件
* 支持自定义行间分隔线
* 支持下拉刷新
* 支持上拉加载

属性

* ItemSeparatorComponent：分割线组件，
* ListFooterComponent：结尾组件
* ListHeaderComponent：头组件
* data：列表数据
* horizontal：设置为true则变为水平列表。
* numColumns：列数组件内元素必须是等高的,无法支持瀑布流布局
* columnWrapperStyle：numColumns大于1时，设置每行的样式
* getItemLayout：如果知道行高可以用此方法节省动态计算行高的开销。
* refreshing：是否正在加载数据
* onRefresh：设置此属性需要一个标准的 RefreshControl 控件，刷新数据
* renderItem：渲染每个组件
* onViewableItemsChanged：当一个新的Item渲染或者隐藏 的时候调用此方法。参数：info: {viewableItems: Array, changed: Array} viewableItems：当前可见的Item，changed：渲染或者隐藏的Item。
* scrollToEnd：滚动到末尾，如果不设置getItemLayout属性的话，可能会比较卡。
* scrollToIndex：滚动到制定的位置
* scrollToOffset：滚动到指定的偏移的位置。

2. Navigation第三方库

下载第三方库

Yarn add react-navigation –save 或者 npm install react-native --save

TabNavigator: Tab导航栏

const Tabbar = createBottomTabNavigator( {路由设置} , {导航栏设置} );





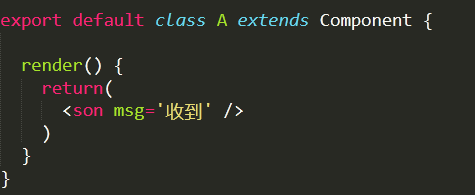
（五）组件之间的通信

1. 嵌套通信

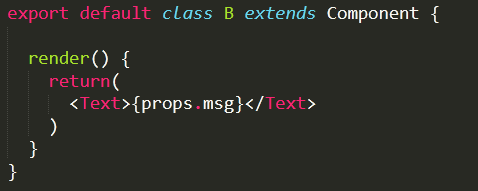
（1）父组件向子组件通信

最常见的通信方式，父组件通过向子组件传递Props，子组件得到Props进行相应处理

父组件App.js：



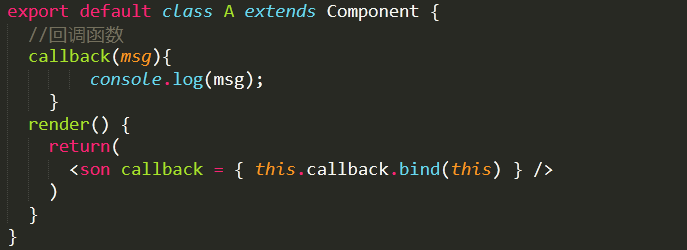
子组件son.js：



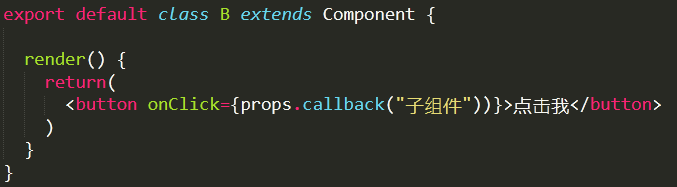
（2）子组件向父组件通信

回调函数

父组件App.js：



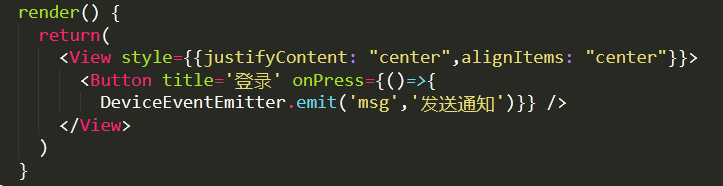
子组件son.js：



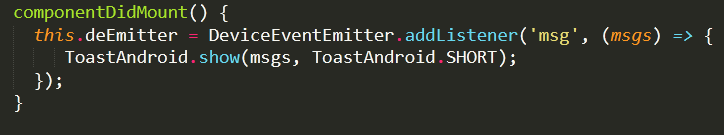
2. 非嵌套通信

（1）通过DeviceEventEmitter，类似于广播

发送方法：



接受方法：



注销



（2）框架Redux

（六）网络Fetch

1.GET

**fetch(this.props.api\_url)**

**.then((response) => response.json())**

**.then((responseData) => {**

**console.log(responseData);**

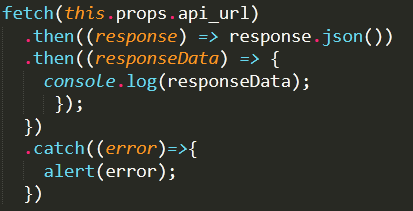
**});**

**})**

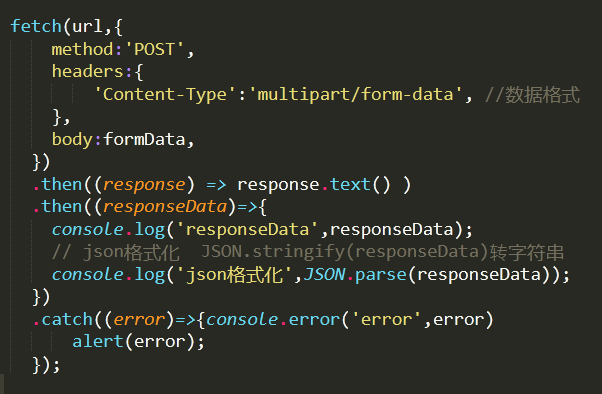
**.catch((error)=>{**

**alert(error);**

**})**



2.POST



**fetch(url,{**

**method:'POST',**

**headers:{**

**'Content-Type':'multipart/form-data', //数据格式**

**},**

**body:formData,**

**})**

**.then((response) => response.text() )**

**.then((responseData)=>{**

**console.log('responseData',responseData);**

**// json格式化 JSON.stringify(responseData)转字符串**

**console.log('json格式化',JSON.parse(responseData));**

**})**

**.catch((error)=>{console.error('error',error)**

**alert(error);**

**});**

1. 集成到原生应用

方法1：

（一）配置 React Native 依赖和项目结构

1.创建项目目录结构

首先创建一个空目录用于存放 React Native 项目，然后在其中创建一个/android子目录，把你现有的 Android 项目拷贝到/android子目录中。

2.创建JS依赖包

（1）根目录创建package.json文件：

{

"name": "ReactNativeApp",

"version": "0.0.1",

"private": true,

"scripts": {

"start": "node node\_modules/react-native/local-cli/cli.js start"

}

}

（2）安装依赖

①安装react-native

nmp install react-native 或者 yarn add react-native

②安装指定版本react

版本号必须一致，版本提示在安装react-native会出现以下语句

warning "react-native@0.52.2" has unmet peer dependency "react@16.2.0".

安装 yarn add react@16.2.0

（二）配置Android工程

（1）配置maven

①添加 React Native 依赖

在你的 app 中 build.gradle 文件中添加 React Native 依赖:

dependencies {

implementation "com.facebook.react:react-native:+"

}

②添加maven

在项目的 build.gradle 文件中为 React Native 添加一个 maven 依赖的入口，必须写在 "allprojects" 代码块中:

repositories {

maven {

url "$rootDir/../node\_modules/react-native/android"

}

}

③配置权限

网络权限

<uses-permission android:name="android.permission.INTERNET" />

开发者菜单

<activity android:name="com.facebook.react.devsupport.DevSettingsActivity" />

3.代码实现

（1）RN

（2）Android

①调用RN界面

**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())**

**.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();**

**}**

**}**

②绑定生命周期

**@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();**

**}**

**}**

③设置返回事件

**@Override**

**public void onBackPressed() {**

**if (mReactInstanceManager != null) {**

**mReactInstanceManager.onBackPressed();**

**} else {**

**super.onBackPressed();**

**}**

**}**

**4.运行程序**

**方法2：**

**1.Activity**

**2.Application**

1. **原生组件和原生模块**

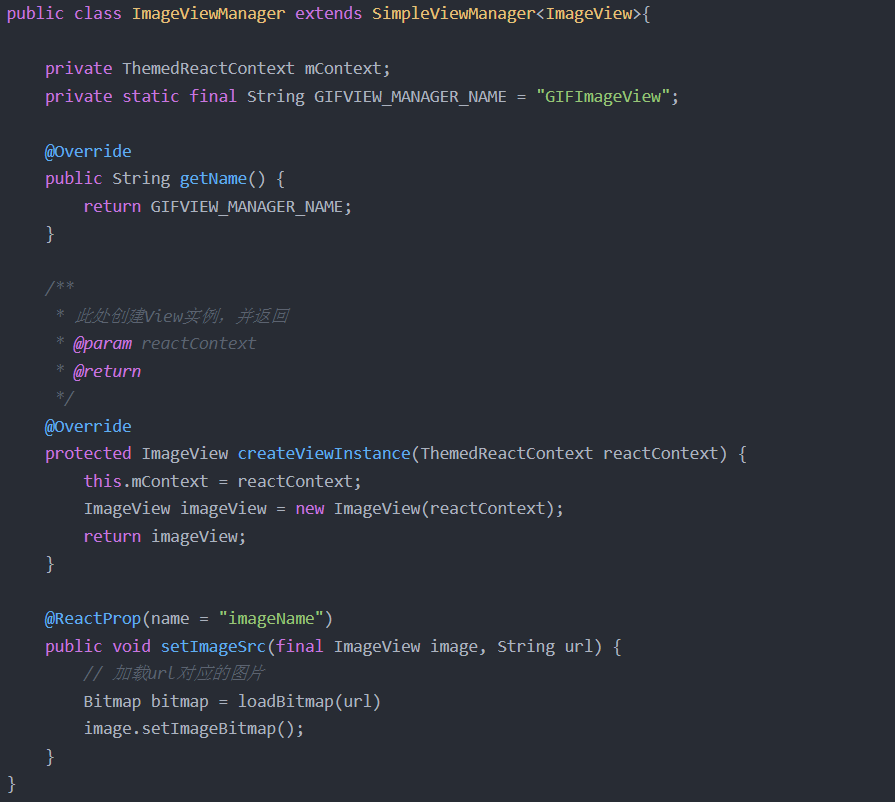
**创建UI模块**

**创建原生UI组件需要我们定义SimpleViewManager的子类，并实现getName、createViewInstance、@ReactProp注解的方法。**

**（1）getName：返回原生UI模块的唯一标识名称。**

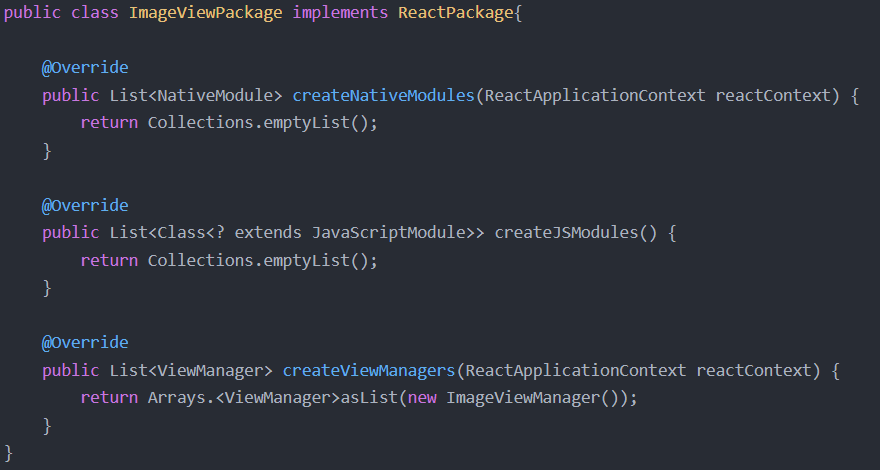
**（2）createViewInstance：创建UI视图实例。**

**（3）@ReactProp：标注要注册的属性。**

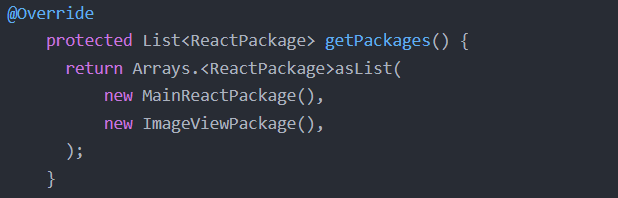


**注册UI模块**

**继承ReactPackage**



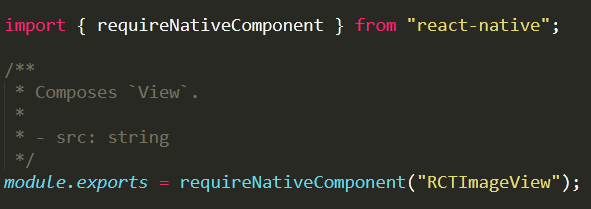
**3.** **将包管理类添加到Application的getPackages中**



**4.RN层调用原生模块**

**创建 JavaScript 模块并且定义 Java 和 JavaScript 之间的接口层**

**RCTImageView.js**



**调用组件**

