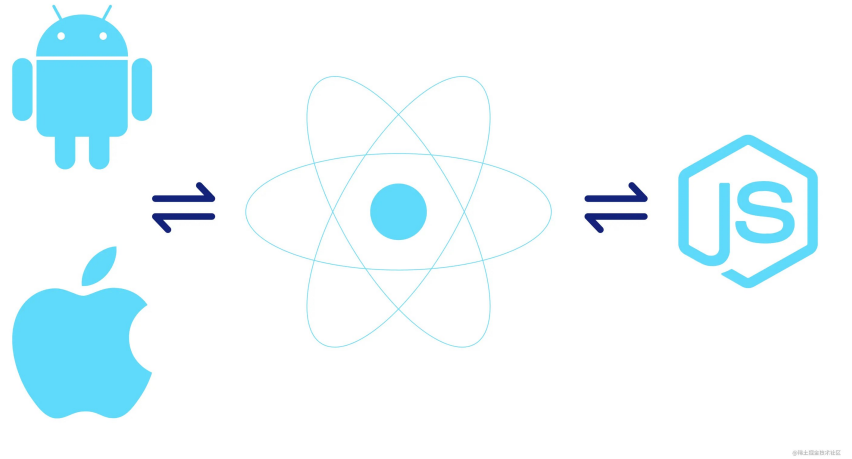
手把手创建 react-native Module

神秘前端游戏开... LV.2
2021年06月18日 23:38 · 阅读 564



> 文中代码均可在 这里 中找到

react-native 最让我觉得舒心的事情，就是能把在 native 端做好的东西包装好，交给 Javascript 端调用。如果你在公司做，自家的产品能一直不倒而且开发人员有着很好的组装意识的话，在产品的的不断迭代过程中，有着各 种的模块包装出来，而这些包装好的模块，可以交给 Javascript 开发人员调用，这样就能把一些功能下放到灵活调用的 react-native 去了。

如果你自己是全栈开发人员，那么自己在原生方面积累下来的代码，就能融合到 react-native 中去。方便二次开发的使用。

那么，在 react-native 下如何编写和原生交流的代码？官网的 教程 已经足够的详细。而这里主要是对自己在开发过程中需要做的流程和遇到的问题一个记录和总结。

> 最好有 java / Object-C 的开发基础

模块和组件（Module / Component）

在官网里涉及 native 开发的，提到最多的就是 Module 和 UI Component。这两个有什么不同呢？简单点理解，UI Component 应该是更加高级的 Module。
当我们需要把 native 的一些方法导出到 javaScript 端的时候，构建一个 Module 是最好的办法。
当我们想把一个 View 的控制权交给 javaScript 端的时候，那就应该创建 UI Component 。
当然如果你想创建的是一个无需 javaScript 端控制的弹层界面，你可以直接使用 Module。

Module

下面就按照在实际开发中的疑惑来逐步了解如何去处理以下这些问题

1. 如何命名
2. 参数如何传入
3. native 层结果的包装和返回
4. 常量导出
5. 如何在 native 代码中获取 (context / Activity & viewController)

iOS

命名

iOS 的模块命名是很简单的事情，宏的存在帮我们省了很多事情。要实现它只需要三个步骤

- 创建一个类，实现 `RCTBridgeModule` 协议
- 使用宏 `RCT_EXPORT_MODULE` 定下 javaScript 能调用的模块名字 (实现 `moduleName` 和 `load` 方法)
- 使用宏 `RCT_EXPORT_METHOD` 导出方法到 javaScript

```object-c
// 创建一个 ToolsManager 的类，实现 RCTBridgeModule 协议
// ToolsManager.h
#import <Foundation/Foundation.h>
#import <React/RCTBridgeModule.h>
NS_ASSUME_NONNULL_BEGIN
@interface ToolsManager : NSObject<RCTBridgeModule>
@end
NS_ASSUME_NONNULL_END

// ToolsManager.m
#import "ToolsManager.h"

@implementation ToolsManager
//+ (NSString *)moduleName {
//  return @"ToolsManager"
//}
//+ (void)load{
//  RCTRegisterModule(self);
//}
// 以上代码等价于
//RCT_EXPORT_MODULE("ToolsManager");
// 以上代码等价于：不传入名字就自动按照当前类名命名，但要注意的是，RCT 开头的就会删除 RCT(RCTToolsManager -> ToolsManage
RCT_EXPORT_MODULE();
// 导出方法 test 到 javaScript
RCT_EXPORT_METHOD(test){
  NSLog(@"testLog");
}

@end
```

那么以上一连串的操作之后，我们在 javaScript 上就能调用 `ToolsManager` 这个模块。可用尝试运行一下它的 `test` 方法

```javascript
// javaScript
import { NativeModules } from 'react-native'
const { ToolsManager } = NativeModules
ToolsManager.test()
```

这时候我们就完成了一个最简单的模块和最简单的模块方法。

传参

上面的方法只是最简单的方法，在实际开发中我们需要传入各种各样的参数，而这些参数应该怎么填写

- 传入普通类型参数
  普通类型的参数其实就 *字符串* 和 *数字* 比较多，boolean 其实也是也是数字转换而来的
  - string (NSString)
  - number (NSInteger, float, double, CGFloat, NSNumber)
  - boolean (BOOL, NSNumber)

```object-c
// ToolsManager.m
RCT_EXPORT_METHOD(testSimple:(NSString *)name number:(float)number flag:(BOOL)flag){
  NSLog(@"testLog %@",name);
  NSLog(@"testLog %f",number);
  NSLog(@"testLog %d",flag);
}
```

```javascript
// javascript
ToolsManager.testSimple("name", 1123, true)
```

- 传入数组和对象
  复杂的类型除了数组和对象之外，还有很多类似 style / date 这些 javaScript 常用的对象，react-native 也提供了一个 `RCTConvert` 来帮我们自动转换,详情可以参考 RCTConvert。但是在日常需求里面，我觉得这个功能并不常用，更多时候我只是传入业务需要的自定义对象。
  - array (NSArray)
  - object (NSDictionary)

```object-c
// ToolsManager.m
RCT_EXPORT_METHOD(testComplex:(NSArray *)array object:(NSDictionary *)dict){
  NSLog(@"testLog %@",array);
  NSLog(@"testLog %@",dict);
}
```

```javascript
// javascript
```

```
ToolsManager.testComplex([1, 2, 3, 4], { a: 123, b: 'das' })
```

- 传入函数
  javascript 用的最多的方法就是回调函数，而在和 native 交互中，获取结果的最好方法就是使用回调。那么关于这个回调函数的使用，我们会到下一节结果返回的时候再讨论。

结果返回

参数的传入是 javaScript 和 native 交流的第一步，而只有结果返回了，整个交流才算跑通。返回结果的方法有两种，前面提及的回调函数，以及 javaScript 上最讨人喜欢的 promise，因为它会你的代码变得更加直观。

- 回调函数
  回调函数的使用是最简单的，其实可以看成 javaScript 传入的函数就是一个 `RCTResponseSenderBlock` 类型参数。
  调用 RCTResponseSenderBlock 的时候，传入的数组，回到 javaScript 端，就都转换成 javaScript 中对应的一个个返回值。(具体类型对应可以看回去 传参 的时候的类型转换)
  需要注意的是，**RCTResponseSenderBlock 只能使用一次**。我们可以存起来使用，但是不能多次使用。

```
Object-C 复制代码
//ToolsManager.m
RCT_EXPORT_METHOD(testCallback:(RCTResponseSenderBlock)callback){
  NSNumber *num  = @1;
  NSString *str = @"string";
  NSArray *arr = @[@"arr1",@"arr2"];
  NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:@"v1",@"k1",@"v2",@"k2", nil];
  callback(@[num,str,arr,dict]);
}
```

```
javaScript 复制代码
//javascript
ToolsManager.testCallback((num: number, str: string, arr: string[], obj: any) => {
      console.log(num, str, arr, obj);
    })
```

- promise
  相对 回调函数 来说，promise 更让人喜欢，而这时候创建 promise 函数则需要用到另一个宏：`RCT_REMAP_METHOD`。和创建函数有所不同的是，这个宏需要传入两个参数：

  - javaScript 端调用的函数名字
  - 真实的 object-c 函数(这个参数最后两个参数分别是 RCTPromiseResolveBlock 类型 和 RCTPromiseRejectBlock 类型，而前面的参数都是对应 javaScript 传入来的参数) 和 回调函数 相同的是，resolve 和 reject 都可以存起来，使用的时候再调用，而且只能调用一次。`RCT_REMAP_METHOD`

```
Object-C 复制代码
//ToolsManager.m
RCT_REMAP_METHOD(testPromise,testPromise:(BOOL)flag withResolver:(RCTPromiseResolveBlock)resolve
                 rejecter:(RCTPromiseRejectBlock)reject){
  if(flag){
    resolve(@"result");
  }
  else{
    NSDictionary *error = [NSDictionary dictionaryWithObjectsAndKeys:@"message",@"error", nil];
    resolve(error);
  }
}
```

```
javaScript 复制代码
//javascript
const testPromise = async () => {
    try {
      const res = await ToolsManager.testPromise(true)
      console.log(res)

    } catch (error) {
      console.log(error.message)
    }

    try {
      const res = await ToolsManager.testPromise(false)
      console.log(res)

    } catch (error) {
      console.log(error.message)
    }
}
```

常量导出

Module 在 javaScript 看来就是一个对象，里面除了我们刚才一直在写的方法之外，还应该有属性存在，那么从 native 端，就应该使用 `constantsToExport` 来输出这个模块的一些静态属性来给 javaScrit 端使用。而按照官方来说，使用了 `constantsToExport` 之后，我们还应该明确告诉 react-native，我们的这个模块是否需要在主线程初始化。

object-c 复制代码

```
- (NSDictionary *)constantsToExport{
  return @{
    @"SUCCESS":@1,
    @"ERROR":@2,
  };
}
// 在主线程初始化
+ (BOOL)requiresMainQueueSetup{
  return YES;
}
```

javaScript 复制代码

```
console.log('ToolsManager.SUCCESS: ', ToolsManager.SUCCESS);
console.log('ToolsManager.ERROR: ', ToolsManager.ERROR);
```

如何获取 viewController (native 中弹层)

传入，返回都能很好的执行之后，我们为了更加深入获取 native 的功能，有时候，我们需要调去一些原生的页面获取一些信息回来，这时候，使用 viewController 就变得很重要了。

object-c 复制代码

```
//ToolsManager.m
RCT_EXPORT_METHOD(testToFile:(RCTResponseSenderBlock)callback){
  // 存起来，后面使用
  self.callback = callback;
  // 获取 rootViewController
  UIViewController *rootViewController = RCTPresentedViewController();
  // 由于和 react-native 和 javascript 通信的不是在 主线程中。
  dispatch_async(dispatch_get_main_queue(), ^{
       UIDocumentPickerViewController *documentPicker = [[UIDocumentPickerViewController alloc] initWithDo
       documentPicker.delegate = self;
       documentPicker.modalPresentationStyle = UIModalPresentationFormSheet;
       [rootViewController presentViewController:documentPicker animated:YES completion:nil];
     });
}
```

Android

命名 / 注册

和 iOS 不一样的是，安卓上，我们除了需要创建一个 Module 类之外，还需要创建一个 Package 类，前者和 iOS 的 Module 类差不多，而后者则用来注册模块的。

- 创建一个 ToolsManager 的类，实现 `ReactContextBaseJavaModule` 接口
- 实现 `getName` 方法，定下 javaScript 能调用的模块名字
- 使用注解 `@ReactMethod`，定义对应的函数，该函数就能导出到 javaScript
- 创建 ToolsPackage 类，实现 `ReactPackage` 方法
- 在 MainApplication 类的 `getPackages` 方法中，添加 packages

Java 复制代码

```
// ToolsManager.java
// 创建一个 ToolsManager 的类，实现 ReactContextBaseJavaModule 接口
class ToolsManager extends ReactContextBaseJavaModule {
    @NonNull
    @Override
    public String getName() {
        return "ToolsManager";
    }
    @ReactMethod
    public void test(){
        Log.d("ToolsManager","test");
    }
}
// ToolsPackage.java
// 创建 ToolsPackage 类，实现 ReactPackage 方法
class ToolsPackage implements ReactPackage {
    @NonNull
    @Override
    public List<NativeModule> createNativeModules(@NonNull ReactApplicationContext reactContext) {
        List<NativeModule> modules = new ArrayList<>();
        modules.add(new ToolsManager());
        return modules;
    }
```

```
    @NonNull
    @Override
    public List<ViewManager> createViewManagers(@NonNull ReactApplicationContext reactContext) {
        return Collections.emptyList();
    }
}

//  MainApplication.java
@Override
protected List<ReactPackage> getPackages() {
  @SuppressWarnings("UnnecessaryLocalVariable")
  List<ReactPackage> packages = new PackageList(this).getPackages();
  packages.add(new ToolsPackage()); // add ToolsPackage
  return packages;
}
```

那么以上一连串的操作之后，我们在 javaScript 上就能调用 `ToolsManager` 这个模块。可用尝试运行一下它的 `test` 方法

javascript 复制代码

```
// javaScript
import { NativeModules } from 'react-native'
const { ToolsManager } = NativeModules
ToolsManager.test()
```

这时候我们就完成了一个最简单的模块和最简单的模块方法。

传参

由于 java 和 javaScript 比较相似，传递参数的时候，也比较直观

- 传入普通类型参数
  - string (String)
  - number (Integer, Double, Float)
  - boolean (Boolean)

Java 复制代码

```
//ToolsManager.java
@ReactMethod
public void testSimple(String name,float number,boolean flag){
    Log.d("ToolsManager",name);
    Log.d("ToolsManager", String.valueOf(number));
    Log.d("ToolsManager", String.valueOf(flag));
}
```

JavaScript 复制代码

```
// JavaScript
ToolsManager.testSimple("name", 1123, true)
```

- 传入数组和对象
  和 iOS 不同，传入数组和对象的时候，react-native 会把它们转换成 `ReadableArray` 和 `ReadableMap`
  - Array (ReadableArray)
  - Object (ReadableMap)

Java 复制代码

```
//ToolsManager.java
@ReactMethod
public void testComplex(ReadableArray array, ReadableMap obj){
    Log.d("ToolsManager", String.valueOf(array));
    Log.d("ToolsManager", String.valueOf(obj));
}
```

JavaScript 复制代码

```
// JavaScript
ToolsManager.testComplex([1, 2, 3, 4], { a: 123, b: 'das' })
```

- 传入函数
  安卓下的传入函数会给转换成 react-native 提供的 `CallBack` 类。具体在结果返回的回调函数中会讲到。

结果返回

- 回调函数 回调函数可以理解成一个 javaScript 传入的函数转换成了一个 `CallBack` 类的值，我们可以把它存起来，在适当的时候调用。但需要注意的是**只能调用一次**
  在类似转换上，我们需要 `WritableArray` 和 `WritableMap` 来包转好数据，这样 javaScript 才能接受正确的 array 和 object

java 复制代码

```java
//ToolsManager.java
@ReactMethod
public void testCallback(Callback callback){
    WritableArray array = Arguments.createArray();
    array.pushString("arr1");
    array.pushString("arr2");
    WritableMap obj = Arguments.createMap();
    obj.putString("k1","v1");
    obj.putString("k2","v2");
    callback.invoke(1,"string",array,obj);
}
```

javaScript 复制代码

```javascript
ToolsManager.testCallback((num: number, str: string, arr: string[], obj: any) => {
  console.log(num, str, arr, obj)
})
```

- Promise
  相对回调函数来说，Pomise 的创建只是为了在函数传递的参数末尾添加一个 `Promise` 类型的参数，我们拿到这个参数之后，也是可以存起来，再适当的时候使用。同样也是**只能调用一次**

Java 复制代码

```java
//ToolsManager.java
@ReactMethod
public void testPromise(Boolean flag,Promise promise){
    if(flag){
        promise.resolve("result");
    }
    else {
        promise.reject("404","error");
    }
}
```

javaScript 复制代码

```javascript
//javaScript
const testPromise = async () => {
    try {
      const res = await ToolsManager.testPromise(true)
      console.log(res)

    } catch (error) {
      console.log(error.message)
    }

    try {
      const res = await ToolsManager.testPromise(false)
      console.log(res)

    } catch (error) {
      console.log(error.message)
    }
}
```

常量导出

除了方法的导出，我们还可以导出一些常量给 javaScript 使用。

- 重写 `getContants` 方法
- 返回一个 `Map<String, Object>` 类型

java 复制代码

```java
//ToolsManager.java
@Nullable
@Override
public Map<String, Object> getConstants() {
    final Map<String, Object> constants = new HashMap<>();
    constants.put("SUCCESS", 1);
    constants.put("ERROR", 0);
    return constants;
}
```

javaScript 复制代码

```javascript
//javaScript
console.log('ToolsManager.SUCCESS: ', ToolsManager.SUCCESS);
console.log('ToolsManager.ERROR: ', ToolsManager.ERROR);
```

如何获取 context / Activity

context 是安卓开发中必不可少的东西。而对于在我们自己创建的 `ToolsManager` 类中如何获取 `context`。其实在创建模块的时候，我们已经找到答案了。
在 ToolsPackage 中 `createNativeModules` 的中，我们能看到一个 `ReactApplicationContext` 类型的参数

被传了过来，而在 `createNativeModules` 中，我们需要 new 一个模块的实例时(`new ToolsManager`) 是否可以把这个 `ReactApplicationContext` 给传进来呢？

java 复制代码

```java
// ToolsManager.java
class ToolsManager extends ReactContextBaseJavaModule {
    ReactApplicationContext reactContext;
    public ToolsManager(@Nullable ReactApplicationContext reactContext) {
        super(reactContext);
        this.reactContext = reactContext;
    }
}
// ToolsPackage.java
@NonNull
@Override
public List<NativeModule> createNativeModules(@NonNull ReactApplicationContext reactContext) {
    List<NativeModule> modules = new ArrayList<>();
    modules.add(new ToolsManager(reactContext));
    return modules;
}
```

在模块中获取 Activity 也很简单，因为在 ReactContextBaseJavaModule 类，早已定义好一个 `getCurrentActivity` 方法供我们使用

java 复制代码

```java
// ToolsManager.java
@ReactMethod
public void testToFile(Callback callback){
    this.callback = callback;
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    Uri uri = Uri.parse("/");
    intent.setDataAndType(uri, "*/*");
    Activity activity = getCurrentActivity();
    if(activity != null){
        activity.startActivity(Intent.createChooser(intent, "Open folder"));
    }
}
```

javascript 复制代码

```javascript
const openFile = () => {
    ToolsManager.testToFile(() => {

    })
}
```

在上面的代码我们使用 `startActivity` 启动了一个文件管理器，而希望接收这个文件管理器 Activity 的返回参数，我们还需要做一些设置：

- 调用 reactContext 的 `addActivityEventListener` 方法
- 实现 `ActivityEventListener` 接口
- 在 onActivityResult 获取结果

java 复制代码

```java
// ToolsManager.java
class ToolsManager extends ReactContextBaseJavaModule implements ActivityEventListener {
    ReactApplicationContext reactContext;
    public ToolsManager(@Nullable ReactApplicationContext reactContext) {
        super(reactContext);
        this.reactContext = reactContext;
        this.reactContext.addActivityEventListener(this);
    }
}
@Override
public void onActivityResult(Activity activity, int requestCode, int resultCode, Intent data) {

}

@Override
public void onNewIntent(Intent intent) {

}
```