# react-native
# 原生扩展

原生模块
原生UI组件

马

富林

# 应用场景

- 需要使用React-Native未封装的原生功能
- 重用已有的原生组件或者第三方组件
- 多线程调用以及高性能要求的功能

# 原生模块扩展

- 创建模块
- 注册模块
- JS端调用
- 回调函数

# 创建模块

- 1:创建ReactContextBaseJavaModule子类
- 2:实现getName()方法
- 3:为提供给JS调用的方法添加注解@ReactMethod

# 创建模块

```
public class JavaModule extends ReactContextBaseJavaModule{

    public JavaModule(ReactApplicationContext reactContext) {
        super(reactContext);
    }

    @Override
    public String getName() {
        return "ModuleName";
    }

    @ReactMethod
    public void method(String attribute){

    }
}
```

# 注册模块

- 1:创建ReactPackage子类
- 2:实现createNativeMadules()方法

- 在MainActivity中为ReactInstanceManger添加ReactPackage子类的对象

# 注册模块

```java
public class NativeReactPackage implements ReactPackage {
    @Override
    public List<NativeModule> createNativeModules(ReactApplicationContext reactContext) {
        List<NativeModule> modules = new ArrayList<>();
        modules.add(new JavaModule(reactContext));
        return modules;
    }
```

```java
public class MainActivity extends TravelActivity implements View.OnClickListener{
    private ReactInstanceManager mReactInstanceManager;
    private ReactRootView mReactRootView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mReactRootView = new ReactRootView(this);
        mReactInstanceManager = ReactInstanceManager.builder()
                .setApplication(getApplication())
                .setBundleAssetName("index.android.bundle")
                .setJSMainModuleName("index.android")
                .addPackage(new MainReactPackage())
                .addPackage(new NativeReactPackage())
                .setUseDeveloperSupport(BuildConfig.DEBUG)
                .setInitialLifecycleState(LifecycleState.RESUMED)
                .build();
```

# JS端调用

- java

```java
public class JavaModule extends ReactContextBaseJavaModule{

    public JavaModule(ReactApplicationContext reactContext) {
        super(reactContext);
    }

    @Override
    public String getName() {
        return "ModuleName";
    }

    @ReactMethod
    public void method(String attribute){

    }
}
```

- js

```javascript
var JavaModule = NativeModules.ModuleName;

JavaModule.method("attribute");
```

# 回调函数

```java
@ReactMethod
public void callbackMethod(String attribute, Callback callback){
    //....
    callback.invoke("callbackValue1","callbackValue2");
}
```

- js

```javascript
var JavaModule = NativeModules.ModuleName;
JavaModule.callbackMethod(
    "attribute",
    (callbackValue1, callbackValue2) => {
        console.log("callbackValue1="+callbackValue1+";callbackValue2="+callbackValue2);
});
```

# 原生UI组件扩展

- 1:创建ViewManager子类
- 2:注册该ViewManager子类
- 3:js端调用

# 创建ViewManager子类

- 1:创建ViewManager子类
- 2:实现getName()方法
- 3:实现createViewInstance()方法
- 4:为提供给JS调用的属性设置方法添加注解@ReactProp

# 创建ViewManager子类

```java
public class NativeViewManager extends SimpleViewManager<NativeView> {

    public static final String REACT_CLASS = "NativeView";

    @Override
    public String getName() {
        return REACT_CLASS;
    }

    @Override
    protected NativeView createViewInstance(ThemedReactContext reactContext) {
        return new NativeView(reactContext);
    }

    @ReactProp(name = "attr")
    public void setAttr(NativeView view, String attr) {
        view.setAttr(attr);
    }
}
```

# 注册ViewManager子类

```
public class NativeReactPackage implements ReactPackage {

    @Override
    public List<NativeModule> createNativeModules(ReactApplicationContext reactContext) {
        List<NativeModule> modules = new ArrayList<>();
        modules.add(new JavaModule(reactContext));
        return modules;
    }

    @Override
    public List<ViewManager> createViewManagers(ReactApplicationContext reactContext) {
        List<ViewManager> viewManagers = new ArrayList<>();
        viewManagers.add(new NativeViewManager());
        return viewManagers;
    }
```

```
public class MainActivity extends TravelActivity implements View.OnClickListener{
    private ReactInstanceManager mReactInstanceManager;
    private ReactRootView mReactRootView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mReactRootView = new ReactRootView(this);
        mReactInstanceManager = ReactInstanceManager.builder()
                .setApplication(getApplication())
                .setBundleAssetName("index.android.bundle")
                .setJSMainModuleName("index.android")
                .addPackage(new MainReactPackage())
                .addPackage(new NativeReactPackage())
                .setUseDeveloperSupport(BuildConfig.DEBUG)
                .setInitialLifecycleState(LifecycleState.RESUMED)
```

# JS端调用

```
var {requireNativeComponent, PropTypes } = require('react-native');

var iface = {
  name: 'NativeView',
  propTypes: {
    attr:PropTypes.string,
  },
};

module.exports = requireNativeComponent('NativeView', iface);
```

```
    render: function () {
        return (

            <NativeView/>

        );
    }
```

# 谢谢