以下是一个利用 **TypeScript** 或 **JavaScript** 的接口注入方法来模拟 React Native 的原生模块的示例。我们通过依赖注入实现对接口的动态替换，以便在测试或开发阶段使用模拟实现。

### 场景：模拟一个原生模块

假设我们有一个原生模块 DeviceModule，用于获取设备的信息，比如电池电量和设备名称。

#### 1. 定义接口

首先，定义一个接口来描述 DeviceModule 的功能：

// DeviceModuleInterface.ts

export interface DeviceModuleInterface {

getBatteryLevel(): Promise<number>;

getDeviceName(): Promise<string>;

}

#### 2. 提供默认实现

在生产环境中，我们使用实际的原生模块：

// DeviceModule.ts

import { NativeModules } from 'react-native';

import { DeviceModuleInterface } from './DeviceModuleInterface';

export class DeviceModule implements DeviceModuleInterface {

async getBatteryLevel(): Promise<number> {

return NativeModules.DeviceModule.getBatteryLevel();

}

async getDeviceName(): Promise<string> {

return NativeModules.DeviceModule.getDeviceName();

}

}

#### 3. 提供一个模拟实现

在测试或开发环境中，我们可以提供一个模拟版本的实现：

// MockDeviceModule.ts

import { DeviceModuleInterface } from './DeviceModuleInterface';

export class MockDeviceModule implements DeviceModuleInterface {

async getBatteryLevel(): Promise<number> {

return Promise.resolve(100); // 模拟满电量

}

async getDeviceName(): Promise<string> {

return Promise.resolve('Mock Device'); // 模拟设备名称

}

}

#### 4. 使用依赖注入

通过将实现注入到业务代码中，我们可以动态切换实现：

// App.tsx

import React, { useState, useEffect } from 'react';

import { Text, View } from 'react-native';

import { DeviceModuleInterface } from './DeviceModuleInterface';

import { DeviceModule } from './DeviceModule';

// import { MockDeviceModule } from './MockDeviceModule'; // 测试环境时替换

interface AppProps {

deviceModule: DeviceModuleInterface;

}

const App: React.FC<AppProps> = ({ deviceModule }) => {

const [batteryLevel, setBatteryLevel] = useState<number | null>(null);

const [deviceName, setDeviceName] = useState<string | null>(null);

useEffect(() => {

deviceModule.getBatteryLevel().then(setBatteryLevel);

deviceModule.getDeviceName().then(setDeviceName);

}, [deviceModule]);

return (

<View>

<Text>Battery Level: {batteryLevel ?? 'Loading...'}</Text>

<Text>Device Name: {deviceName ?? 'Loading...'}</Text>

</View>

);

};

// 在生产环境

export default () => <App deviceModule={new DeviceModule()} />;

// 在测试环境

// export default () => <App deviceModule={new MockDeviceModule()} />;

### 优点

1. **灵活性**：通过注入，可以在运行时动态切换实际实现和模拟实现。
2. **可测试性**：在测试中注入模拟实现，减少对实际原生模块的依赖。
3. **代码解耦**：业务逻辑与底层实现分离，便于维护和扩展。

此方法适用于任何需要切换依赖实现的场景，尤其适合 React Native 开发中的原生模块模拟。