## 什么是React Native？它与React有什么区别？

React Native是一个用于构建移动应用程序的开源框架。它基于React，允许开发人员使用JavaScript和React的语法来构建原生移动应用。

### 区别：

#### ‌应用场景‌：

‌React‌：主要用于构建Web应用程序，特别是单页面应用（SPA）。它基于Web开发，以浏览器为渲染平台‌。

‌React Native‌：专注于构建原生移动应用，支持iOS和Android两大平台。它允许开发者使用同一套代码同时为iOS和Android开发应用‌。

#### ‌渲染机制‌：

‌React‌：通过虚拟DOM技术，将React组件的状态变化映射到真实的DOM上，实现高效的用户界面更新‌。

‌React Native‌：直接将React组件映射为原生平台的UI组件（如iOS的UIView、Android的View），实现接近原生应用的性能和外观‌。

#### ‌组件和API‌：

‌React‌：使用标准的HTML标签和CSS样式。

‌React Native‌：提供了一套特定的组件（如<View>、<Text>、<Image>等）和样式API，这些组件和API会根据目标平台的不同而有所差异‌。

#### ‌性能‌：

‌React‌：虽然通过虚拟DOM优化了UI更新过程，但在性能上仍不如原生应用。

‌React Native‌：通过使用原生组件，提供了接近原生应用的性能和流畅的用户体验‌。

#### ‌开发工具‌：

‌React‌：开发者通常使用浏览器上的开发者工具进行调试和测试。

‌React Native‌：可以使用类似于Xcode和Android Studio的集成开发环境（IDE）进行调试和测试‌。

#### ‌社区支持‌：

‌React‌：拥有庞大的开发者社区和丰富的第三方库，为开发者提供了强大的支持。

‌React Native‌：虽然社区支持不如React广泛，但也在不断发展和壮大。‌

## React Native相对于原生的ios和Android有哪些优势？

1.性能媲美原生APP   
2.使用JavaScript编码，只要学习这一种语言

3.绝大部分代码安卓和IOS都能共用

4.组件式开发，代码重用性很高

5.跟编写网页一般，修改代码后即可自动刷新，不需要慢慢编译，节省很多编译等待时间

6.支持APP热更新，更新无需重新安装APP

缺点： 内存占用相对较高 版本还不稳定，一直在更新，现在还没有推出稳定的1.0版本

1 优势

性能方面媲美原生App。

绝大部分代码同时适用IOS/Android，一套代码两套系统适用。

使用Javascript编码，上手容易。

组件式开发，易于管理维护，代码复用率高。

代码更改后会自动刷新，节省等待时间。

支持热更新，更新无需重新安装App。

2 劣势

RN组件库不全，第三方组件库也不全，当遇到某些特殊功能，需要花费大量时间、精力完成；性能方面也无法媲美原生，还是会有一些损耗，特别是大数据交换时；

系统适配方面， IOS版本略好，android发展较慢；

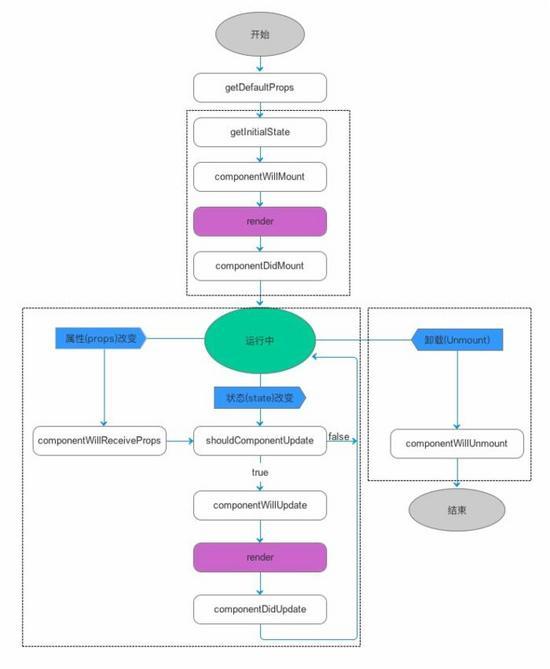
编程方面， ios和android代码并非通用，有可能需要维护两套代码或者在代码中做一些条件判断或编译；

开发人员还是需要会原生开发，不然自定义组件无法编码；

开发复杂应用必须精通原生开发，开发效率并不比原生开发的熟手快。很多问题（包括兼容性问题解决）任然需要原生开发。

升级RN版本或需要大动干戈，尤其向下兼容不好；

## React Native组件的生命周期



## 当你调用setState的时候，发生了什么事？

当调用 setState 时，React会做的第一件事情是将传递给 setState 的对象合并到组件的当前状态。 这将启动一个称为和解（reconciliation）的过程。 和解（reconciliation）的最终目标是以最有效的方式，根据这个新的状态来更新UI。 为此，React将构建一个新的 React 元素树（您可以将其视为 UI 的对象表示）。 一旦有了这个树，为了弄清 UI 如何响应新的状态而改变，React 会将这个新树与上一个元素树相比较（ diff ）。 通过这样做， React 将会知道发生的确切变化，并且通过了解发生什么变化，只需在绝对必要的情况下进行更新即可最小化 UI 的占用空间。

## props和state相同点和不同点

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。

不同点  
1.初始值来源：state的初始值来自于自身的getInitalState（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。

## shouldComponentUpdate 应该做什么

其实这个问题也是跟reconciliation有关系。  
“和解（ reconciliation ）的最终目标是以最有效的方式，根据新的状态更新用户界面”。  
如果我们知道我们的用户界面（UI）的某一部分不会改变，  
那么没有理由让 React 很麻烦地试图去弄清楚它是否应该渲染。  
通过从 shouldComponentUpdate 返回 false，  
React 将假定当前组件及其所有子组件将保持与当前组件相同

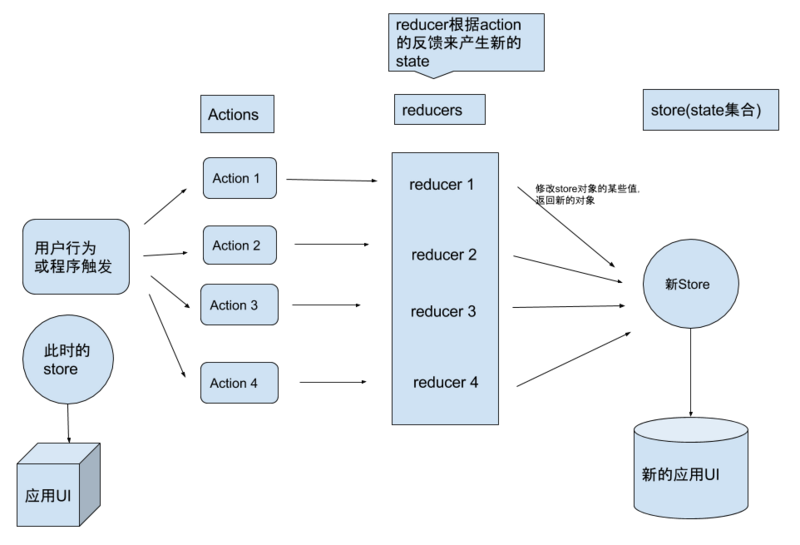
## reactJS的props.children.map函数来遍历会收到异常提示，为什么？应该如何遍历？

this.props.children 的值有三种可能：

1.当前组件没有子节点，它就是 undefined;2.有一个子节点，数据类型是 object ；3.有多个子节点，数据类型就是 array 。

系统提供React.Children.map()方法安全的遍历子节点对象

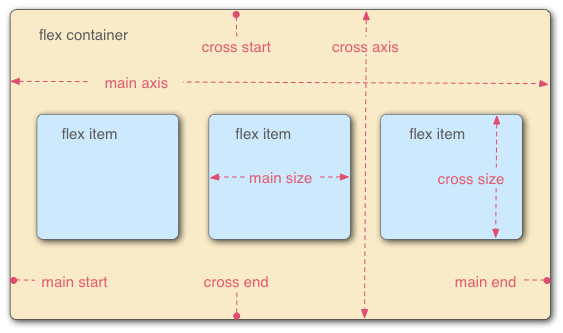
## redux状态管理的流程

  
action是用户触发或程序触发的一个普通对象。  
reducer是根据action操作来做出不同的数据响应，返回一个新的state。  
store的最终值就是由reducer的值来确定的。（一个store是一个对象, reducer会改变store中的某些值）  
action -> reducer -> 新store -> 反馈到UI上有所改变。

## 加载bundle的机制

要实现RN的脚本热更新，我们要搞明白RN是如何去加载脚本的。 在编写业务逻辑的时候，我们会有许多个js文件，打包的时候RN会将这些个js文件打包成一个叫index.android.bundle(ios的是index.ios.bundle)的文件，所有的js代码(包括rn源代码、第三方库、业务逻辑的代码)都在这一个文件里，启动App时会第一时间加载bundle文件，所以脚本热更新要做的事情就是替换掉这个bundle文件。

## Flex布局

采用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成员，称为Flex项目（flex item），简称"项目"。  


容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做main start，结束位置叫做main end；交叉轴的开始位置叫做cross start，结束位置叫做cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做main size，占据的交叉轴空间叫做cross size。

容器的属性  
以下6个属性设置在容器上。  
flex-direction 属性决定主轴的方向（即项目的排列方向)。  
flex-wrap 属性定义，如果一条轴线排不下，如何换行。  
flex-flow flex-flow属性是flex-direction属性和flex-wrap属性的简写形式。  
justify-content 定义了项目在主轴上的对齐方式。  
align-items 属性定义项目在交叉轴上如何对齐。  
align-content align-content属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

## 请简述 code push 的原理

code push 调用 react native 的打包命令，将当前环境的非 native 代码全量打包成一个 bundle 文件，然后上传到微软云服务器（Windows Azure）。 在 app 中启动页（或 splash 页）编写请求更新的代码（请求包含了本地版本，hashCode、appToken 等信息），微软服务端对比本地 js bundle 版本和微软服务器的版本，如果本地版本低，就下载新的 js bundle 下来后实现更新(code push 框架实现)。

## Redux中同步 action 与异步 action 最大的区别是什么

同步只返回一个普通 action 对象。而异步操作中途会返回一个 promise 函数。当然在 promise 函数处理完毕后也会返回一个普通 action 对象。thunk 中间件就是判断如果返回的是函数，则不传导给 reducer，直到检测到是普通 action 对象，才交由 reducer 处理。

## **如何实现底部TabBar的高度不一样呢？（类似新浪微博底部加号）**

简答：主要考察flex布局绝对定位问题

在React Native中，如果你想实现底部TabBar的高度不同，你可以使用react-navigation库中的**TabBarBottom**组件，并**通过自定义样式来设置不同的高度**。

以下是一个简单的例子，展示如何自定义TabBarBottom的高度

import React from 'react';

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

import { TabBarBottom, TabNavigator } from 'react-navigation';

// 定义一个自定义的TabBar组件

class CustomTabBar extends React.Component {

render() {

return (

<View style={[styles.tabBarContainer, {height: 80}]}>

<TabBarBottom

{...this.props}

renderTabBar={() => <View />} // 不显示默认的TabBar

style={styles.customTabBar}

/>

</View>

);

}

}

// 创建TabNavigator

const MyApp = TabNavigator({

Home: {

screen: HomeScreen,

navigationOptions: {

tabBarLabel: 'Home',

tabBarIcon: ({ tintColor }) => (

<View style={[styles.tabBarIcon, {backgroundColor: tintColor}]}>

<Text style={styles.tabBarIconText}>♡</Text>

</View>

),

},

},

Notifications: {

screen: NotificationsScreen,

navigationOptions: {

tabBarLabel: 'Notifications',

tabBarIcon: ({ tintColor }) => (

<View style={[styles.tabBarIcon, {backgroundColor: tintColor}]}>

<Text style={styles.tabBarIconText}>⚡</Text>

</View>

),

},

},

}, {

tabBarComponent: CustomTabBar,

tabBarPosition: 'bottom',

animationEnabled: false,

swipeEnabled: false,

});

const styles = StyleSheet.create({

tabBarContainer: {

flex: 1,

backgroundColor: 'white',

},

customTabBar: {

backgroundColor: 'blue', // 设置TabBar的背景颜色

},

tabBarIcon: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

width: 50,

height: 50,

},

tabBarIconText: {

color: 'white',

fontSize: 20,

},

});

export default MyApp;

在这个例子中，我们创建了一个CustomTabBar类，在render方法中，我们使用一个View组件包裹TabBarBottom，并通过样式{height: 80}来设置自定义的高度。TabBarBottom组件的style属性用于设置TabBar的背景样式。

## 请您简单介绍一下redux？dva? mobx？

简答：redux ==> action/reducer/store

mobx ==>数据双向绑定

## React中Element 和 Component 有何区别？

简单地说，一个 React element 描述了你想在屏幕上看到什么。

换个说法就是，一个 React element 是一些 UI 的对象表示。

一个 React Component 是一个函数或一个类，

它可以接受输入并返回一个 React element

(通常是通过 JSX ，它被转化成一个 createElement 调用）。

## 描述事件在React中的处理方式

为了解决跨浏览器兼容性问题，

您的 React 中的事件处理程序将传递 SyntheticEvent 的实例，

它是 React 的浏览器本机事件的跨浏览器包装器。

这些 SyntheticEvent 与您习惯的原生事件具有相同的接口，除了它们在所有浏览器中都兼容。

有趣的是，React 实际上并没有将事件附加到子节点本身。

React 将使用单个事件监听器监听顶层的所有事件。

这对于性能是有好处的，这也意味着在更新DOM时，React 不需要担心跟踪事件监听器

## 什么是react-navigation?

react-navigation是致力于解决导航卡顿，数据传递，Tabbar和navigator布局.

（1）StackNavigator：用来跳转页面和传递参数

（2）TabNavigator：类似底部导航栏，用来在同一屏幕下切换不同界面

（3）DrawerNavigator：侧滑菜单导航栏，用于轻松设置带抽屉导航的屏幕

### createStackNavigator

就是一种基于栈的路由管理方式,栈的特点就是先入后出，最新入栈的界面会显示在最顶部，这也是Android管理Activity的方式，也是React-Native App打开页面最主要的方式。

### createSwitchNavigator

switch，意思也比较明显，就是选择的意思。也就是说，当你使用这个路由时，内存中只会存在一个页面或者一个路由(多路由情况)。其实，大多数App都有一个欢迎界面，这个界面在App中只会显示一次，如果单单是使用栈的形式，不好控制出栈的操作，实现起来就比较复杂，那么我们的createSwitchNavigator就能派上用场了。当跳转到我们的主路由的时候，欢迎界面也就消失了。

### createBottomTabNavigator

如果我们要实现类似微信首页多tab多界面的时候，createBottomTabNavigator就能派上用场了，他通过单个路由管理多个tab.

### createMaterialTopTabNavigator

同过实现一个material风格的顶部选择导航，相对于createBottomTabNavigator，有些属性不同。

ReactNative是实现原生开发的框架

可以使用react的语法和js来编写  
开发理念: Learn once,write anywhere

## 运行阶段生命周期调用顺序

componentWillReceiveProps–>shouldComponentUpdate --> componentWillupdate --> componentDidUpdate

## JS调用原生方法

IOS

RCTBridgeModule RCT\_EXPORT\_MODULE RCT\_EXPORT\_METHOD RCT\_REMAP\_METHOD Promises Callbacks

android

ReactContextBaseJavaModule ReactMethod MyReactPackage里增加模块

NativeModules.MyNativeModule.callNativeMethod(‘成功调用原生方法’);

## react native redux中间件

react-redux redux-actions redux-promise redux-thunk redux-logger redux-devtools

## 请解释React Native的工作原理。

React Native通过将JavaScript代码解释执行，并使用原生组件渲染用户界面。

React Native应用程序包含两个主要线程：JavaScript线程和原生UI线程。

JavaScript线程负责执行JavaScript代码，处理业务逻辑和用户交互。它使用Bridge将JavaScript和原生代码之间进行通信。

原生UI线程负责处理界面渲染和用户输入响应。它使用原生组件和API来构建用户界面，并将界面渲染到设备屏幕上。

通过Bridge，JavaScript线程可以发送命令和数据给原生UI线程，并接收来自原生代码的事件和响应。

这种架构使得React Native应用程序能够同时享受JavaScript的灵活性和原生应用的性能。

## 什么是JSX？它在React Native中的作用是什么？

JSX是一种JavaScript的语法扩展，允许在JavaScript代码中编写类似HTML的结构。

在React Native中，JSX用于描述用户界面的结构和组件的层次结构。

JSX使得编写和理解React组件更加直观和易读，同时也提供了一些特性，如组件嵌套、属性传递和事件处理。

## 请解释React Native中的Props和State的区别。

Props（属性）是从父组件传递给子组件的数据，用于配置组件的行为和外观。Props是只读的，子组件不能直接修改它们。

State（状态）是组件的内部可变数据，用于跟踪和控制组件的状态变化。State只能在组件内部使用，并通过setState方法来更新。

主要区别：

Props是从父组件传递给子组件的，State是组件内部的数据。

Props是只读的，State是可变的。

Props用于配置组件，State用于跟踪组件的状态变化。

## 如何在React Native中处理性能优化和内存管理？

在React Native中进行性能优化和内存管理的一些常见方法包括：

避免不必要的重新渲染：使用shouldComponentUpdate或React.memo来避免无需的组件重新渲染。

使用虚拟化列表：对于大型列表或网格，使用虚拟化列表组件（如FlatList）来只渲染可见区域的内容。

使用PureComponent或React.memo：这些组件可以自动执行浅比较来避免不必要的重新渲染。

使用useCallback和useMemo：这些钩子函数可以缓存函数和计算结果，避免在每次渲染时重新创建。

避免内存泄漏：确保在组件卸载时取消订阅事件、清除定时器和释放其他资源。

使用性能分析工具：React Native提供了一些性能分析工具（如PerformanceMonitor），可以帮助识别性能瓶颈和优化机会。

使用React Native Community的第三方库：React Native Community提供了一些优化库，如react-native-fast-image用于高性能的图像加载，react-native-reanimated用于流畅的动画效果等。

## FlatList & SectionList & ScrollView & ListView

ScrollView会把所有子元素一次性全部渲染出来。使用上最简单。但是如果你有一个特别长的列表需要显示，可能会需要好几屏的高度。这时就会占用很大的内存去创建和渲染那些屏幕以外的JS组件和原生视图，性能上也会有所拖累。

ListView 更适用于长列表数据。它会惰性渲染子元素，并不会立即渲染所有元素，而是优先渲染屏幕上可见的元素。

FlatList 是0.43版本开始新出的改进版的ListView，性能更优，但是官方说现在可能不够稳定，尚待时间考验。但是它不能够分组/类/区（section）。

SectionList 也是0.43版本推出的，高性能的分组列表组件。

### FlatList

用于显示一个垂直的滚动列表，其中样式相同仅数据不同

更适于长列表数据，且元素个数可以增删。和ScrollView不同的是，FlatList并不立即渲染所有元素，而是优先渲染屏幕上可见的元素

组件必须的两个属性是data和renderItem。

data是列表的数据源

renderItem则从数据源中逐个解析数据，然后返回一个设定好格式的组件来渲染

**FlatList支持以下功能**

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。

支持跳转到指定行（ScrollToIndex）

### SectionList

SectionList的使用与FlatList类似，不同的是其可以用来渲染二维数据。

SectionList常用属性基本与FlatList相同，不同的是它的数据传入不是使用data，而是使用sections，并且它还特有的分组头部渲染方法renderSectionHeader用于渲染每个分组的头部。