React Naitve 学习

1. 课前准备

1. 环境搭建 React Native中文网
2. 开发工具 前端开发软件：Visual Studio Code、WebStorm；

   移动端开发软件：Xcode、Android Studio
3. 知识储备

   NodeJS React Es6,Es7

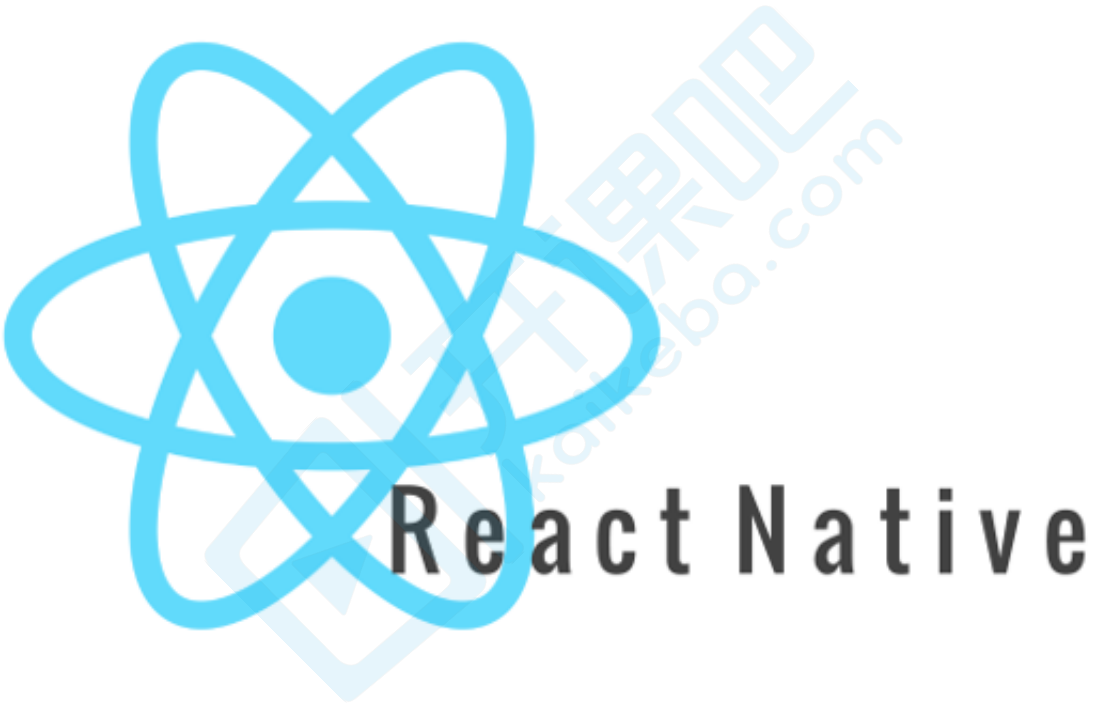
2. 课程大纲

- 环境搭建
- 项目启动与调试
- React Native Flex布局与样式
- 核心组件&&API

课堂目标

- 结合文档成功搭建RN开发环境，结合开发者菜单功能，使用chrome工具调试项目
- 掌握React Native的核心组件和API
- 掌握React Native布局方式

React Native介绍



React Naitve的简介：Facebook在React.js Conf2015大会上推出的一个用于开发Android和iOS App的一个框架，主要编程语言是JavaScript。它的出现使用**即拥有Native的用户体验，又保留React的开发效率**。

在 React Native 出现前，我们通常会选择这三种移动技术（Native App、HTML5、Hybrid）之一进行开发。

- `Native App`：开发原生应用自然性能最好，功能强大。但多平台版本的开发、维护要花费大量的人力物力(iOS版本迭代审核需要时间)。
- `HTML5`：虽然有 Web 的优势，即灵活的布局能力、免发版的敏捷迭代潜力、优秀的跨平台特性。在新闻资讯等一些强排版、弱交互的展示类 App 上大展拳脚。但由于 WebView 在移动设备上的性能制约，始终难成大器。
- `Hybrid App`：JS+Native两者相互调用为主，从开发层面实现“一次开发，多处运行”的机制，成为真正适合跨平台的开发。Hybrid App兼具了Native App良好用户体验的优势，也兼具了Web App

使用HTML5跨平台开发低成本的优势，但是这个方法存在诸多问题：无法访问离线数据、无法访问设备、无法远程更新。

- `React Native`：底层引擎是 JavaScript Core，但调用的是原生的组件而非 HTML5 组件。这样运行时可以做到与 Navive App 相媲美的性能体验，同时因为 JavaScript 代码可以使用后端强大的 Web 方式管理，既可以做到高效开发，也可以实现快速部署和问题热修复。

React Native优缺点：

- 优点

1. 跨平台开发：运用React Native，我们可以使用同一份业务逻辑核心代码来创建原生应用运行在Web端，Android端和iOS端；
2. 追求极致的用户体验：实时热部署；
3. learn once,write everywhere：React Native不强求一份原生代码支持多个平台，所以不是write once,run anywhere；

- 缺点

1. react native在iOS上仅支持`iOS7`以上，Android仅支持`Android4.1`以上；
2. 开发成本较高；
3. 部分复杂的界面和操作，RN无法实现(可以考虑原生+React Native混合开发)；

- React Native vs Flutter vs Weex

比较内容	Flutter	Weex	React Native	Android Native
平台实现	通过Dart虚拟机编译成机器码	1. Vue编写的Web页面编译成JS bundle； 2. Native端解析DOM，生成真实的Native控件； 3. Android平台通过ART虚拟机编译成机器码。	1.React编写JS文件，如果是UI界面，会映射到Virtual DOM； 2.通过C++编写的Bridge调用原生的API，控件则是根据DOM映射到原生的View； 3. Android平台通过ART虚拟机编译成机器码。	通过ART虚拟机编译成机器码
绘制引擎	Skia engine	1. 2D绘制：JSCore + Skia engine 2. 3D绘制：JSCore + OpenGL ES	1. 2D绘制：JS V8 + Skia engine 2. 3D绘制：JS V8 + OpenGL ES	1. 2D绘制：Skia engine 2. 3D绘制：OpenGL ES
核心语言	Dart	Vue	React	Java/Kotlin
APK大小	16M	18M	15M	10M
上手难度	一般	容易	较难	/
框架程度	重	较轻	较重	一般
社区	丰富、谷歌团队支持	较小、阿里支持（目前托管Apache）	活跃、Facebook支持	庞大、谷歌团队支持
支持	Android、IOS、Web	Android、IOS、Web	Android、IOS	只支持Android
软件发布	支持热更新	支持热更新	支持热更新	支持热更新
未来前景	1. 谷歌打算推出的新系统Fuchsia采用Flutter； 2. 社区活跃，文档完善； 3. 版本更新迭代很快，基本两天更新一次； 4. 有上线的手机App。	1. 前景不明，阿里自己的产品正在逐渐去Weex化，采用Hybird或ReactNative代替； 2. 后期维护力度小，文档混乱； 3. 版本更新较慢。	1. 成熟度高，已经有很多成功案例； 2. 版本有持续更新，文档完善； 3. 受限于本身跨平台机制，上限不高。	由于目前各种跨平台框架都有缺陷、不够成熟，加上性能方面的优势，Native开发暂时还是主流方式。
性能	自带绘制引擎，号称接近原生，但目前就实际效果看，还有一些距离	因为多了一层JS解析，渲染慢一些	跟Weex差不多	性能表现最佳
TV交互支持	不太友好，需要开发者做大量的焦点处理和按键分发工作，有文档但资料较少	不太友好，需要开发者做大量的焦点处理和按键分发工作，没有文档	明确支持TV交互，有相关文档	本身支持TV平台，5.0版本开始，谷歌大力支持android tv，开发者处理焦点问题和按键事件比较容易
动画	满足现有的动画需求	缺少多页面跳转时的过场动画	满足现有的动画需求	满足现有的动画需求
适用场景	适用跟系统交互少，页面不太复杂的场景	适用于电商类App、业务场景不复杂、需要动态更新的场景	适合业务不太复杂、页面简单的小项目	适用跟系统高度耦合，追求高性能的场景
免安装	不支持	可以支持，需要实现类似快应用的框架（国内只有小程序和快应用支持免安装）	可以支持，需要实现类似快应用的框架（国内只有小程序和快应用支持免安装）	Android的Instant App支持免安装，但是国内用不了
初步结论	综合上面对比数据及开发Demo实践得出如下结论： 1. TV应用由于焦点及按键事件等处理较特殊，基本所有跨平台方案都没有对其支持，并且业务离不开跟底层系统的大量交互，涉及到各个平台兼容问题，开发人员技术栈限制，目前使用原生开发效率更高，效果更好； 2. 谷歌强推Flutter，即将推出的Fuchsia新系统打算统一桌面端、移动端和网页端，并且会采用Flutter框架，以此来推广新系统，值得关注； 3. Weex最大的优势是入门简单，前端和移动端开发都容易上手，代码结构比较清晰，但目前版本迭代较慢，已贡献给Apache，其原班开发人员已不再维护，使用该框架，存在一定风险； 4. ReactNative成熟度高，明确支持TV端交互，体系已经完善，但是上手门槛高，开发难度大，目前在经历大的重构，跟Native控件耦合度高，天花板低，可以小范围使用； 基于以上考虑，TV应用建议使用原生开发，手机应用建议关注Flutter。			

构建项目

1. 在相应的路径下执行命令行：`react-native init 项目名` (名称不可使用连接符等特殊字符,命名可以参考APP应用名称 比如 FaceBook)
2. 跳转到对应路径下执行相应的移动端项目：

```
cd 项目名
react-native run-ios or react-native run-android
```

如果正常，运行效果如下：

RN调试技巧

Developer Menu

Developer Menu是React Native给开发者定制的一个开发者菜单，来帮助开发者调试React Native应用。

在模拟器上开启Developer Menu

Android模拟器：

可以通过`Command⌘ + M`快捷键来快速打开Developer Menu。也可以通过模拟器上的菜单键来打开。

> 心得：高版本的模拟器通常没有菜单键的，不过Nexus S上是有菜单键的，如果想使用菜单键，可以创建一个Nexus S的模拟器。

iOS模拟器：

可以通过`Command⌘ + D`快捷键来快速打开Developer Menu。

Reload

`Reload`选项，单击`Reload`让React Native重新加载js。对于iOS模拟器你也可以通过`Command⌘ + R`快捷键来加载js，对于Android模拟器可以通过双击`r`键来加载js。

> 提示：如果`Command⌘ + R`无法使你的iOS模拟器加载js，则可以通过选中Hardware menu中Keyboard选项下的 “Connect Hardware Keyboard” 。

Enable Live Reload

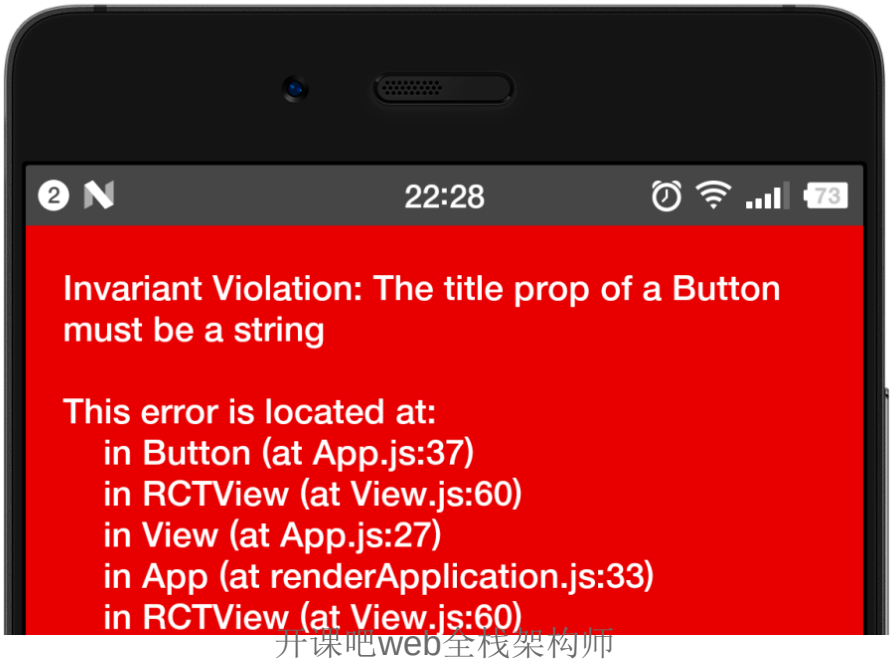
该选项提供了React Native动态加载的功能。当你的js代码发生变化后，React Native会自动生成bundle然后传输到模拟器或手机上

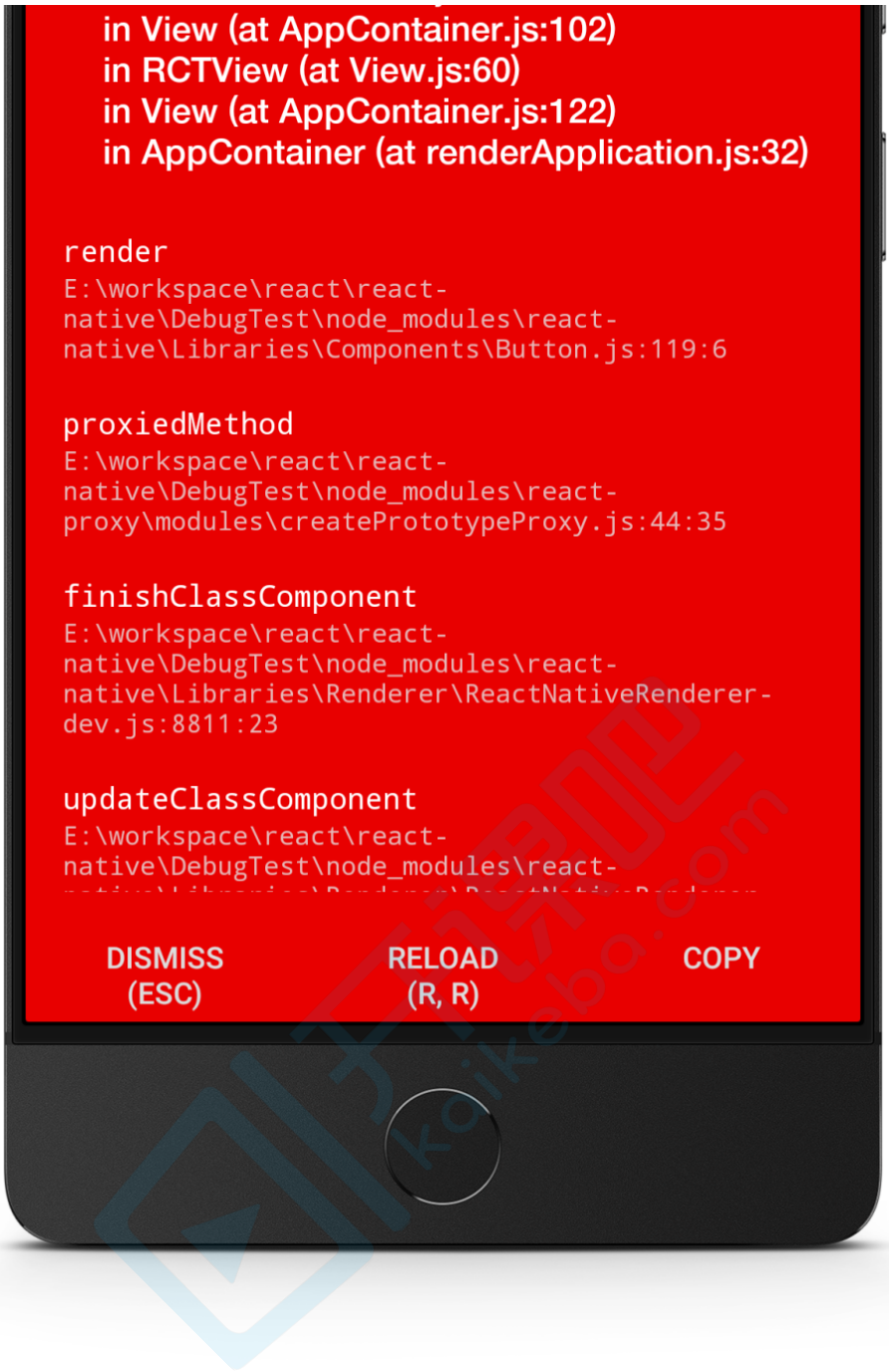
Errors and Warnings

在development模式下，js部分的Errors 和 Warnings会直接打印在手机或模拟器屏幕上，以红屏和黄屏展示。

Errors

React Native程序运行时出现的Errors会被直接显示在屏幕上，以红色的背景显示，并会打印出错误信息。 你也可以通过`console.error()`来手动触发Errors。





Warnings

React Native程序运行时出现的Warnings也会被直接显示在屏幕上，以黄色的背景显示，并会打印出警告信息。 你也可以通过`console.warn()`来手动触发Warnings。 你也可以通过`console.disableYellowBox = true`来手动禁用Warnings的显示，或者通过`console.ignoredYellowBox = ['Warning: ...'];`来忽略相应的Warning

Chrome Developer Tools

第一步：启动远程调试

在Developer Menu下单击”Debug JS Remotely” 启动JS远程调试功能。此时Chrome会被打开，同时会创建一个“http://localhost:8081/debugger-ui.” Tab页。

第二步：打开Chrome开发者工具

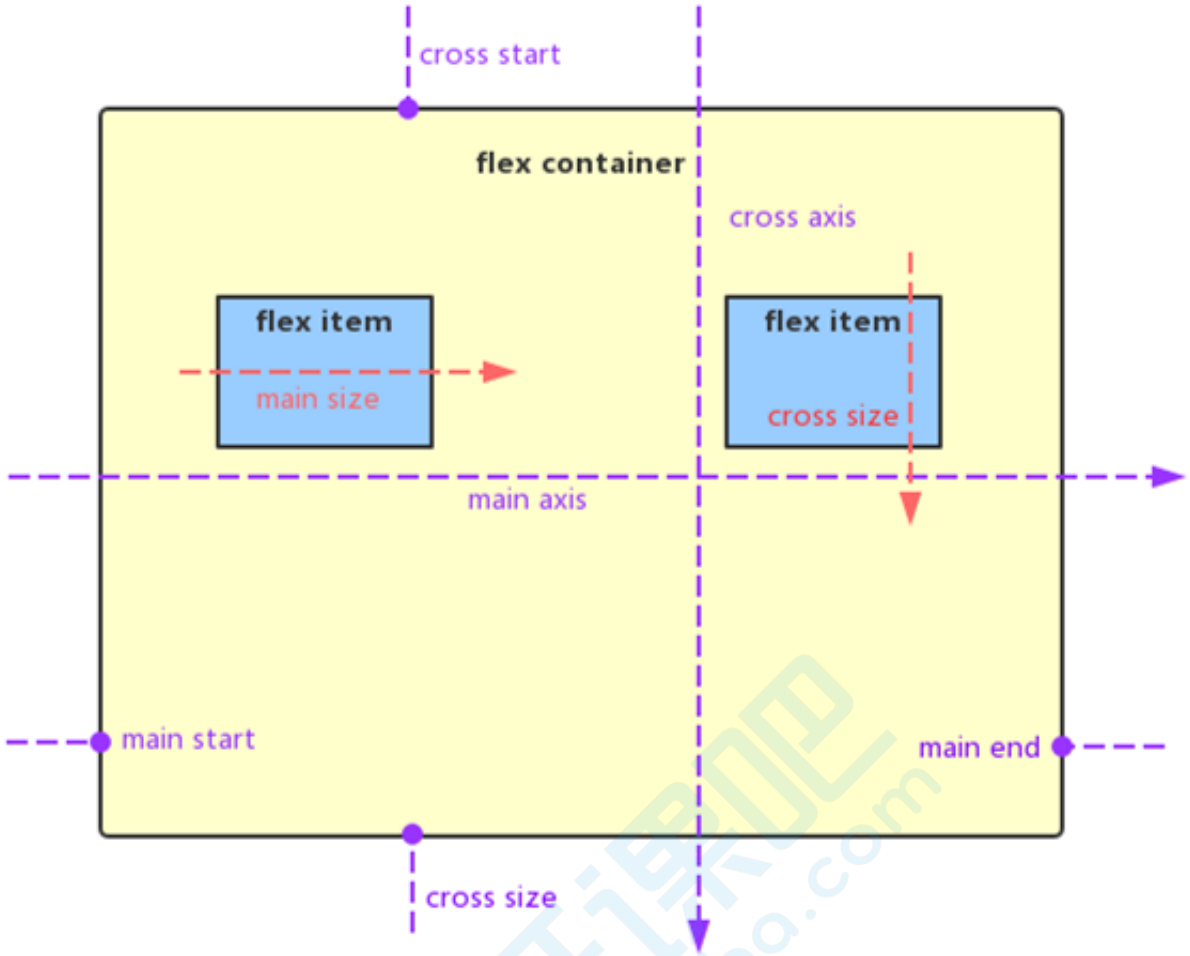
在该“http://localhost:8081/debugger-ui.”Tab页下打开开发者工具。打开Chrome菜单->选择更多工具->选择开发者工具。你也可以通过快捷键(Command⌘ + Option⌥ + I on Mac, Ctrl + Shift + I on Windows)打开开发者工具。

- 断点调试

RN布局与样式

一款好的App离不开漂亮的布局，RN中的布局方式采用的是FlexBox(弹性布局)

FlexBox提供了在不通尺寸设备上都能保持一致的布局方式



宽和高

在学习FlexBox之前首先要清楚一个概念"宽和高"。一个组件的高度和宽度决定了他在屏幕上的尺寸，也就是大小

像素无关

在RN中尺寸是没有单位的，它代表的是设备独立像素

```
<View style={{width:100,height:100,margin:10,backgroundColor:'gray'}}>
  <Text style={{fontSize:16,margin:20}}>尺寸</Text>
</View>
```

上述代码，运行在Android上时，View的长宽被解释成：100dp 100dp，字体被解释成16sp，运行在ios上时尺寸单位被解释成pt,这些单位确保了布局在任何不通DPI的手机屏幕上，显示效果一致

和而不同

RN中FlexBox和Web Css上FlexBox工作方式是一样的，但有些地方还是有出入的

flexDirection:

RN中默认是flexDirection:'column',Web Css中默认是 flex-direction:'row'

alignItems:

RN中默认alignItems: 'stretch',在Web Css中默认 align-items:'flex-start'

flex:

RN中只接受一个属性，Web css 可以接受多个属性：flex: 2 2 10%

不支持的属性： align-content flex-basis order flex-flow flex-grow flex-shrink

Flex in RN

以下属性是RN所支持的Flex属性

容器属性

flexDirection: row | column| row-reverse | column-reverse

flexWrap: wrap | noWrap //换行

justifyContent: flex-start | flex-end | center | space-between | space-around

alignItems: flex-start | flex-end | center | stretch

项目属性

alignSelf

auto(default) 元素继承了父容器的align-item属性，如果没有则为'stretch'

stretch

center

flex-start

flex-end

flex:定义了一个元素可伸缩的能力，默认是0

样式

在RN中样式 需要引入StyleSheet API

写法一：

```
<View style={styles.container}></View>
const styles = StyleSheet.create({
  container:{
    ...
  }
});
```

组件内写法：

```
<View style={{backgroundColor:'red'}}></View>
//or
<View style={[styles.container,{backgroundCorlor:'red'}]}></View>
```

RN核心组件与API

在RN中使用原生组件，是依赖React的，所以在使用过程中需要导入react

```
import React, { Component } from "react";
import { Button, Platform, StyleSheet, Text, View } from "react-native";
```

常用组件介绍

- **Button**：一个简单的跨平台的按钮组件。可以进行一些简单的定制。

```
<Button
  onPress={onPressLearnMore} //用户点击此按钮时所调用的处理函数
  title="Learn More" //按钮按钮内显示的文本
  color="#841584" //文本的颜色(iOS)，或是按钮的背景色(Android)
  disabled={false} //按钮是否可以点击
  accessibilityLabel="Learn more about this purple button" //用于给残障人士显示的
文本（比如读屏应用可能会读取这一内容
/>
```

- **ActivityIndicator**：显示一个圆形的 loading 提示符号。

```
<View style={[styles.container, styles.horizontal]}>
  <ActivityIndicator
    size="large" //指示器的大小，默认为'small'[enum('small', 'large'), number]。目前只能在 Android 上设定具体的数值
    animating={true} //是否要显示指示器动画，默认为 true 表示显示，false 则隐藏。
    hidesWhenStopped={false} //在animating为 false 的时候，是否要隐藏指示器（默认为true）。如果animating和hidesWhenStopped都为 false，则显示一个静止的指示器。
    color="#0000ff" //滚轮的前景颜色（默认为灰色）。
  />
</View>
```

- **Image**：用于显示多种不同类型图片的 React 组件，包括网络图片、静态资源、临时的本地图片、以及本地磁盘上的图片（如相册）等。

  下面的例子分别演示了如何显示从本地缓存、网络甚至是以`'data:'`的 base64 uri 形式提供的图片。

```
<Image
  source={require('/react-native/img/favicon.png')}
/>
<Image
  style={{width: 50, height: 50}}
  //网络和 base64 数据的图片需要手动指定尺寸
  source={{uri: 'https://facebook.github.io/react-native/docs/assets/favicon.png'}}
/>
<Image
  style={{width: 66, height: 58}}
  //网络和 base64 数据的图片需要手动指定尺寸
  source={{uri:
''}}
/>
```

在 Android 上支持 GIF 和 WebP 格式图片

默认情况下 Android 是不支持 GIF 和 WebP 格式的。你需要在`android/app/build.gradle`文件中根据需要手动添加以下模块：

```
dependencies {
  // 如果你需要支持Android4.0(API level 14)之前的版本
  compile 'com.facebook.fresco:animated-base-support:1.10.0'

  // 如果你需要支持GIF动图
  compile 'com.facebook.fresco:animated-gif:1.10.0'
```

```
  // 如果你需要支持WebP格式，包括WebP动图
  compile 'com.facebook.fresco:animated-webp:1.10.0'
  compile 'com.facebook.fresco:webpsupport:1.10.0'

  // 如果只需要支持WebP格式而不需要动图
  compile 'com.facebook.fresco:webpsupport:1.10.0'
}
```

- **SafeAreaView**：`SafeAreaView`的目的是在一个“安全”的可视区域内渲染内容。具体来说就是因为目前有 iPhone X 这样的带有“刘海”的全面屏设备，所以需要避免内容渲染到不可见的“刘海”范围内。本组件目前仅支持 iOS 设备以及 iOS 11 或更高版本。

  `SafeAreaView`会自动根据系统的各种导航栏、工具栏等预留出空间来渲染内部内容。更重要的是，它还会考虑到设备屏幕的局限，比如屏幕四周的圆角或是顶部中间不可显示的“刘海”区域。

- **Text**：一个用于显示文本的React组件，并且它也支持嵌套、样式，以及触摸处理，在Text内部的元素不再使用flexbox布局，而是采用文本布局。这意味着`<Text>`内部的元素不再是一个个矩形，而可能会在行末进行折叠

```
<Text
ellipsizeMode={"tail"} //这个属性通常和下面的 numberOfLines 属性配合使用,文本超出numberOfLines设定的行数时，截取方式：head- 从文本内容头部截取显示省略号。例如："...efg"，middle - 在文本内容中间截取显示省略号。例如： "ab...yz"，tail - 从文本内容尾部截取显示省略号。例如： "abcd..."，clip - 不显示省略号，直接从尾部截断。
numberOfLines={1} //配合ellipsizeMode设置行数
onPress={} //点击事件
selectable={true}//决定用户是否可以长按选择文本，以便复制和粘贴。
>
</Text>
```

- **TextInput**：是一个允许用户在应用中通过键盘输入文本的基本组件。本组件的属性提供了多种特性的配置，譬如自动完成、自动大小写、占位文字，以及多种不同的键盘类型（如纯数字键盘）,TextInput`在安卓上默认有一个底边框，同时会有一些padding。如果要想使其看起来和iOS上尽量一致，则需要设置`padding: 0

```
<TextInput
  style={{
    width: 100,
    height: 40,
    borderWidth: 3,
    borderColor: "blue"
  }}
  keyboardType={"default"} //决定弹出何种软键盘类型，譬如numeric（纯数字键盘),default,number-pad,decimal-pad,numeric,email-address,phone-pad
```

```
  maxLength={20} //限制文本框中最多的字符数。使用这个属性而不用JS逻辑去实现，可以避免闪烁的现象。
  editable={true} //如果为false，文本框是不可编辑的。默认值为true。
  defaultValue={"xxxx"} //提供一个文本框中的初始值
  caretHidden={true} //如果为true，则隐藏光标。默认值为false。
  autoCapitalize={"none"} //控制TextInput是否要自动将特定字符切换为大写:characters:所有的字符,words：每个单词的第一个字符,sentences：每句话的第一个字符（默认）,none：不切换。
  //当文本框内容变化时调用此回调函数。改变后的文字内容会作为参数传递。从TextInput里取值这就是目前唯一的做法!
  onChangeText={text => {
    this.setState({
      text: text
    });
  }}
  />
```

- **View**：类似于html中的div，容器组件，可以使用[,]的形式返回多个兄弟组件
- **WebView**：`WebView` 创建一个原生的 WebView，可以用于访问一个网页。

```
class MyWeb extends Component {
  render() {
    return (
      <WebView
        source={{uri: 'https://github.com/facebook/react-native'}}
        style={{marginTop: 20}}
        />
    );
  }
```

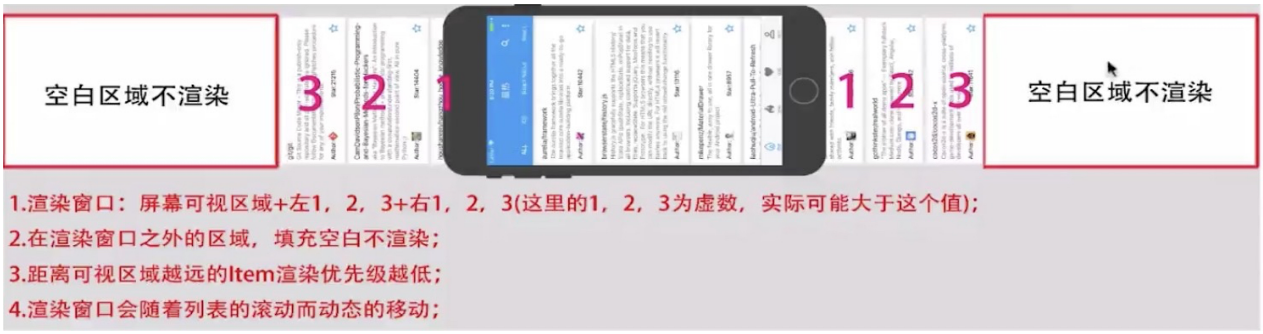
- **ListView**:经常使用ListView的同学都知道，这个组件的性能比较差，尤其是当有大量的数据需要展示的时候，ListView对内存的占用较多，常出现丢帧卡顿现象

  ListView底层实现，渲染组件Item是全量渲染，而且没有复用机制，这就不可避免的当渲染较大数据量时，会发现以下情况：

  - 第一次打开与切换Tab时会出现卡顿或白屏的情况，比如ListView中有100个Item，只能等这100条Item都渲染完成，ListView中的内容才会展示
  - 滑动列表时会出现卡顿不跟手，listVIew渲染大量数据，需要占用较多的内存用于计算

  **未来有很大可能性会被移除

- **VirtualizedList**：`FlatList`和`SectionList`的底层实现，VirtualizedList通过维护一个有限的渲染窗口(其中包含可见的元素)，并将渲染窗口之外的元素全部用合适的定长空白空间代替的方式，极大的改善了内存使用，提高了大量数据情况下的渲染性能。这个渲染窗口能响应滚动行为，元素离可视区越远优先级越低，越近优先级越高，当用户滑动速度过快时，会出现短暂空白的情况。



- **FlatList**：在RN0.43版本中引入了FlatList，SectionList与VirtualizedList，其中VirtualizedList是FlatList和SectionList的底层实现。

  缺点：（1）为了优化内存占用同时保持滑动的流畅，列表内容会在屏幕外异步绘制。这意味着如果用户滑动的速度超过渲染的速度，则会先看到空白的内容。（2）不支持分组列表

```
<FlatList
data={[{key: 'a'}, {key: 'b'}]}
renderItem={({item}) => <Text>{item.key}</Text>}
/>
```

  可以看出跟之前的ListView很像，但是其中少了dataSource，这里，我们只需要传递数据，其它的都交给FlatList处理好了。

  属性与方法详细见FLatList文档

- **RefreshControl**：这一组件可以用在ScrollView或FlatList内部，为其添加下拉刷新的功能。当ScrollView处于竖直方向的起点位置（scrollY: 0），此时下拉会触发一个`onRefresh`事件
- **SwipeableFlatList**：侧滑效果列表组件，在RN0.50版本中引入了SwipeableFlatList，官方文档还没有这个介绍
- **SectionList**：高性能的分组列表组件

  缺点：同样会有空白内容的情况
- TouchableHightlight**：高亮触摸。用户点击时，会产生高亮效果；
- **TouchableOpacity**：透明触摸。用户点击时，被点击的组件会出现透明效果；

常用API介绍

- Dimensions：用于获取设备屏幕的宽高

```
let {height, width} = Dimensions.get('window');
```

- Platform :平台API判断

```
import { Platform, StyleSheet } from "react-native";
const styles = StyleSheet.create({
  height: Platform.OS === "ios" ? 200 : 100
```

```
});

// Platform.select(), 以Platform.OS为 key，从传入的对象中返回对应平台的值：
const Component = Platform.select({
  ios: () => require("ComponentIOS"),
  android: () => require("ComponentAndroid")
})();

// 检测Adr版本
if (Platform.Version === 25) {
  console.log("Running on Nougat!");
}

// 检测iOS版本
const majorVersionIOS = parseInt(Platform.Version, 10);
if (majorVersionIOS <= 9) {
  console.log("Work around a change in behavior");
}

// 当不同平台代码逻辑较为复杂时，可以使用平台扩展名

BigButton.ios.js
BigButton.android.js
const BigButton = require("./BigButton");
```