

安装Homebrew软件包管理工具：

```
ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

安装npm和node.js

安装watchman（非必须）：用于监控bug文件和文件变化；并且可以触发指定的操作

```
brew install watchman
```

安装flow（非必须）：flow是JavaScript的静态文件检查器，方便找出代码只能够存在的类

```
brew install flow
```

安装react-native

```
npm install -g yarn react-native-cli
```

初始化react-native项目

```
react-native init smileyqqReactnative
```

关于Android版本

安装android-sdk

```
brew cask install android-sdk 或者 brew install android-sdk
```

配置.bash_profile（mac中）

```
cd ~
```

```
vim ~/.bash_profile 添加 android-sdk 的路径 export  
ANDROID_HOME=/usr/local/share/android-sdk
```

```
source 该文件 source ~/.bash_profile
```

可以 echo \$ANDROID_HOME 查看是否配置好

安装genymotion， android第三方模拟器

<https://www.genymotion.com/fun-zone/>

<https://www.genymotion.com/download/>

ios上刷新cmd+R

React_native版本管理

```
react-native --version
```

```
npm update -g react-native-cli
```

查询react-native的npm包的最新版本

```
npm info react-native
```

升级或者降级版本

```
npm install --save react-native@版本号
```

二、View组件中常见的属性

React Native组件View，其作用等同于iOS中的UIView，Android中的android.view，或是网页中的<div>标签，它是所有组件的父组件。

Flexbox 弹性布局

 **Transforms** 动画属性

backfaceVisibility enum('visible', 'hidden') 定义界面翻转的时候是否可见

backgroundColor color

borderBottomColor color

borderBottomLeftRadius number

borderBottomRightRadius number

borderBottomWidth number

borderColor color

borderLeftColor color

borderLeftWidth number

borderRadius number

borderRightColor color

borderRightWidth number

borderStyle enum('solid', 'dotted', 'dashed')

borderTopColor color

borderRightWidth number
borderStyle enum('solid', 'dotted', 'dashed')
borderTopColor color
borderTopLeftRadius number
borderTopRightRadius number
borderTopWidth number
borderWidth number
opacity number 设置透明度，取值从0-1；
overflow enum('visible', 'hidden') 设置内容超出容器部分是显示还是隐藏；
elevation number 高度 设置Z轴，可产生立体效果。

移动端着合格做背景颜色很好看 #F5FCFF

Flexbox布局

Flexbox弹性盒子模型

- 浮动布局
- 各种机型屏幕的适配
- 水平和垂直居中
- 自动分配宽度

css中常规布局是基于块和内联流

三、Flexbox的常用属性

3.1 容器属性

a) **flexDirection: `row | row-reverse | column | column-reverse`**

该属性决定主轴的方向（即项目的排列方向）。

row: 主轴为水平方向，起点在左端。

row-reverse: 主轴为水平方向，起点在右端。

column(默认值): 主轴为垂直方向，起点在上沿。

column-reverse: 主轴为垂直方向，起点在下沿。

b) `justifyContent: flex-start | flex-end | center | space-between | space-around`

定义了伸缩项目在主轴线的对齐方式

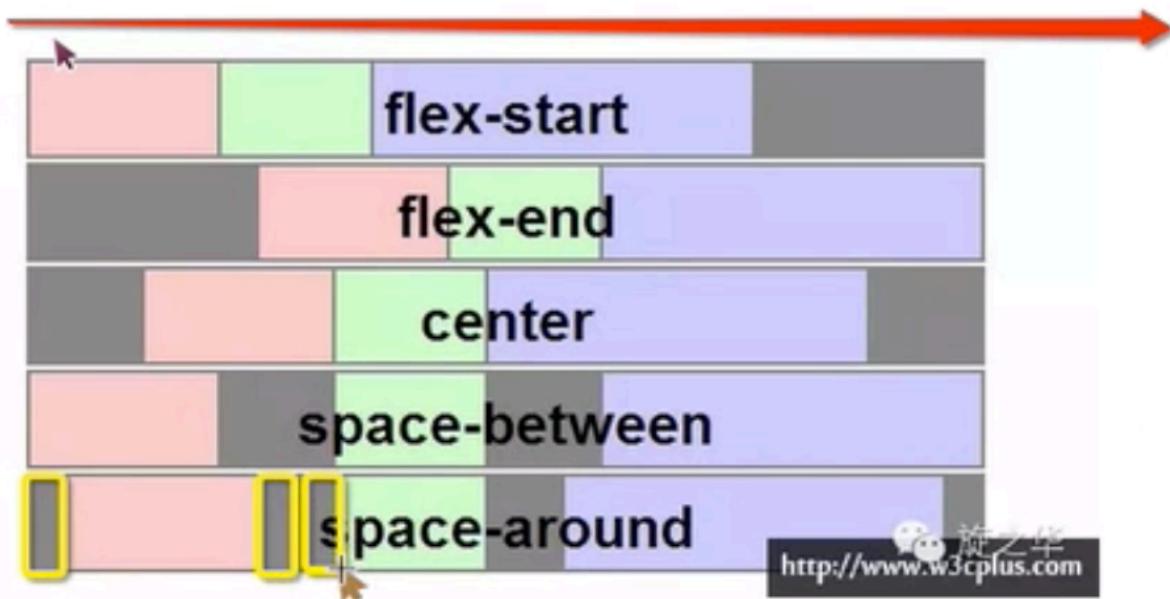
`flex-start`(默认值): 伸缩项目向一行的起始位置靠齐。

`flex-end`: 伸缩项目向一行的结束位置靠齐。

`center`: 伸缩项目向一行的中间位置靠齐。

`space-between`: 两端对齐，项目之间的间隔都相等。

`space-around`: 伸缩项目会平均地分布在行里，两端保留一半的空间。



c) `alignItems: flex-start | flex-end | center | baseline | stretch`

定义项目在交叉轴上如何对齐，可以把其想像成侧轴（垂直于主轴）的“对齐方式”。

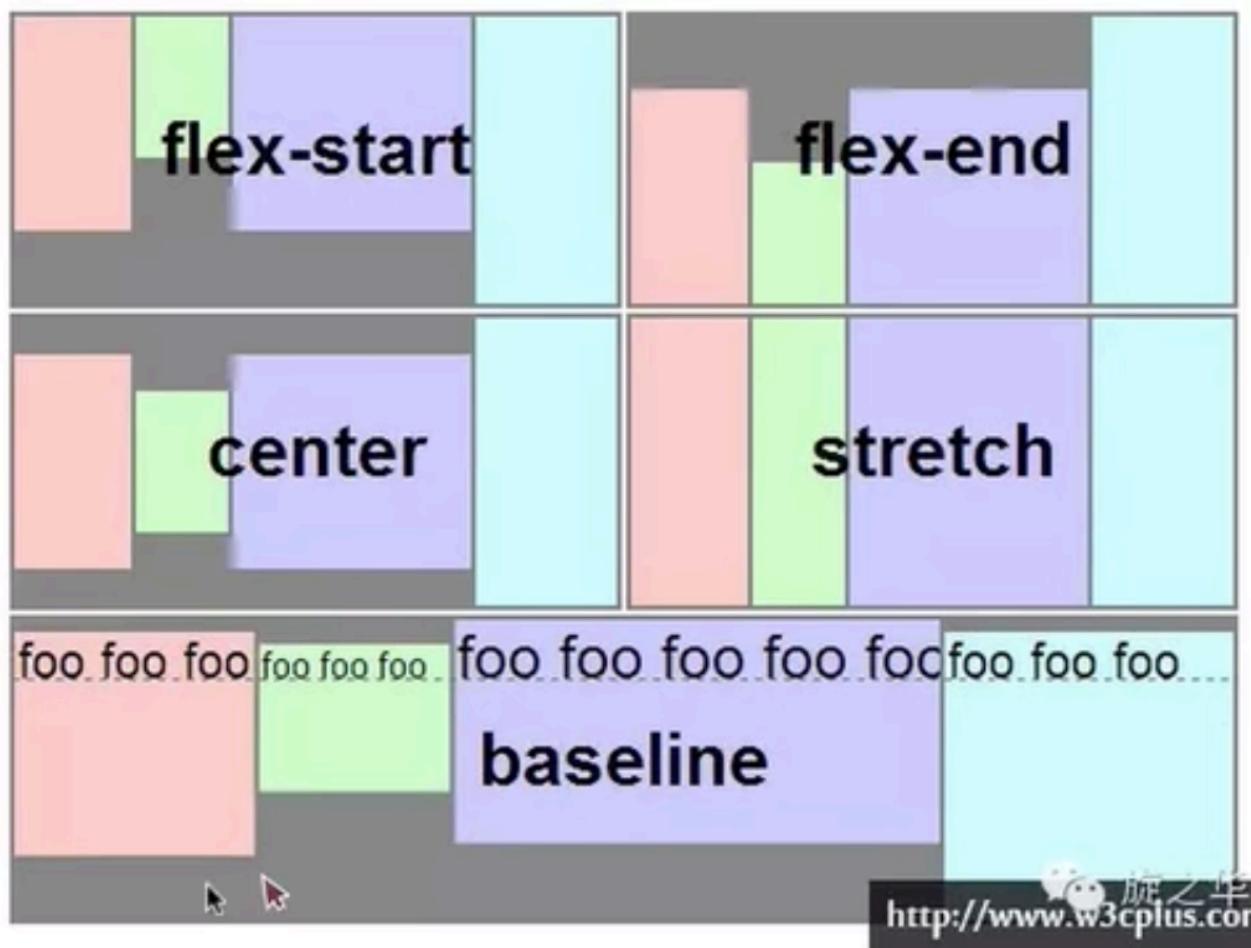
`flex-start`: 交叉轴的起点对齐。

`flex-end`: 交叉轴的终点对齐。

`center`: 交叉轴的中点对齐。

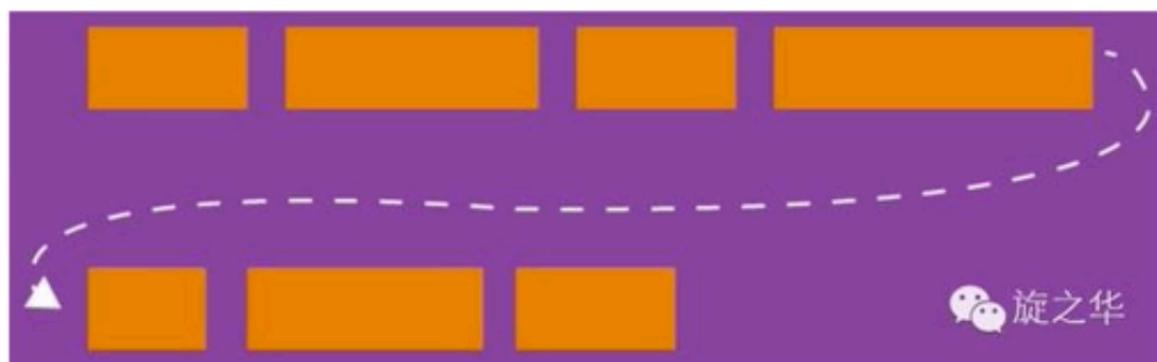
`baseline`: 项目的第一行文字的基线对齐。

`stretch` (默认值) : 如果项目未设置高度或设为auto, 将占满整个容器的高度。



d) flexWrap: `nowrap | wrap | wrap-reverse`

默认情况下，项目都排在一条线（又称“轴线”）上。flex-wrap属性定义，如果一条轴线排不下，如何换行。



```
const styles = StyleSheet.create({
  container: {
    backgroundColor: 'purple',
    // 上边距
    marginTop: 25,
    // 改变主轴的方向
    flexDirection: 'row',
    // 设置主轴的对齐方式
    justifyContent: 'space-around',
    // 设置侧轴的对齐方式
    alignItems: 'center'
  }
});
```

nowrap(默认值): 不换行。



wrap: 换行, 第一行在上方。



wrap-reverse: 换行, 第一行在下方。 (和wrap相反)



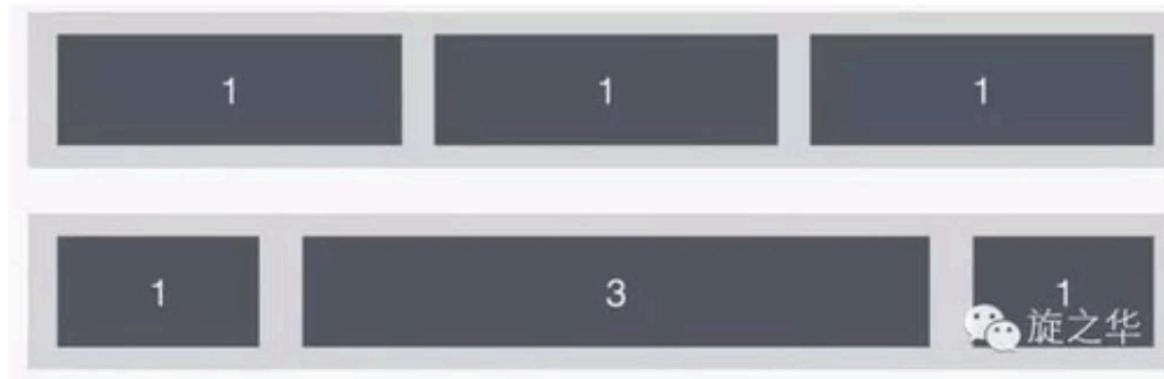
3.2 元素属性

a) flex

“flex-grow”、“flex-shrink”和“flex-basis”三个属性的缩写，其中第二个和第三个参数(flex-shrink、flex-basis)是可选参数。

默认值为“0 1 auto”。

宽度 = 弹性宽度 * (flexGrow / sum(flexGrow))

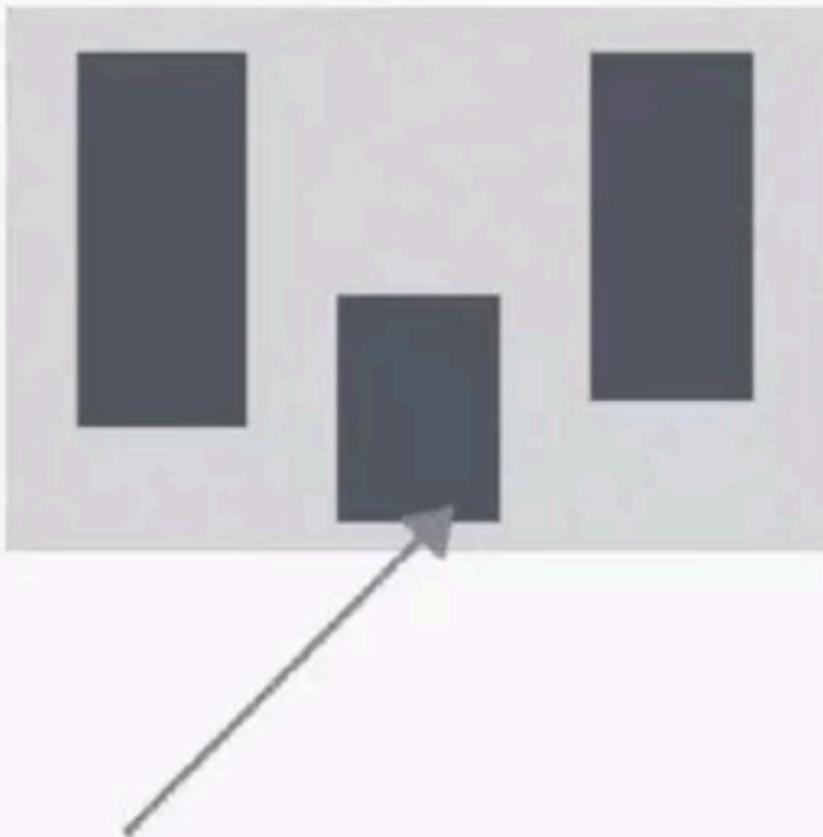


b) alignSelf: “auto | flex-start | flex-end | center | baseline | stretch”

align-self属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。

!image-20190622154358061](/Users/yqp/Library/Application Support/typora-user-images/image-20190622154358061.png)

flex-start



alignSelf: flex-end

获取当前屏幕的宽度高度以及分辨率

```
var Dimensions = require('Dimensions');
export default class App extends Component {
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>当前窗口的宽度:</Text>
        {Dimensions.get('window').width}</Text>
        <Text style={styles.instructions}>当前窗口的高度:</Text>
        {Dimensions.get('window').height}</Text>
        <Text style={styles.instructions}>当前窗口的分辨率:</Text>
        {Dimensions.get('window').scale}</Text>
      </View>
    );
  }
}
```

React Native常用组件之Image

2016-03-13 旋之华 旋之华

一、前言

在开发中还有一个非常重要的组件Image，通过这个组件可以展示各种各样的图片，而且在React Native中该组件可以通过多种方式加载图片资源。

二、Image组件的基本用法

2.1 从当前项目中加载图片

```
<View style={styles.container}>
  <Text style={styles.textMarginTop}>加载本地的图片</Text>
  <Image source={require('./img/2.png')} style={{width: 120, height: 120}} />
</View>
```

该图片资源文件的查找和JS模块相似，该会根据填写的图片路径相对于当前的js文件路径进行搜索。

此外，React Native的Packager会根据平台选择相应的文件，例如:my_icon.ios.png和my_icon.android.png两个文件(命名方式android或者ios)，会分别根据android或者ios平台选择相应的文件。

2.2 加载使用APP中的图片

```
<View style={styles.container}>
  <Text style={styles.textMarginTop}>加载Xcode中的图片</Text>
  <Image source={require('image!icon_homepage_map')} style={{width: 50,height:50}}/>
</View>
```

使用已经打包在APP中的图片资源(例如:xcode asset文件夹以及Android drawable文件夹)

2.3 加载来自网络的图片

客户端的很多图片资源基本上都是实时通过网络进行获取的，写法和上面的加载本地资源图片的方式不太一样，**这边一定需要指定图片的尺寸大小**，实现如下：

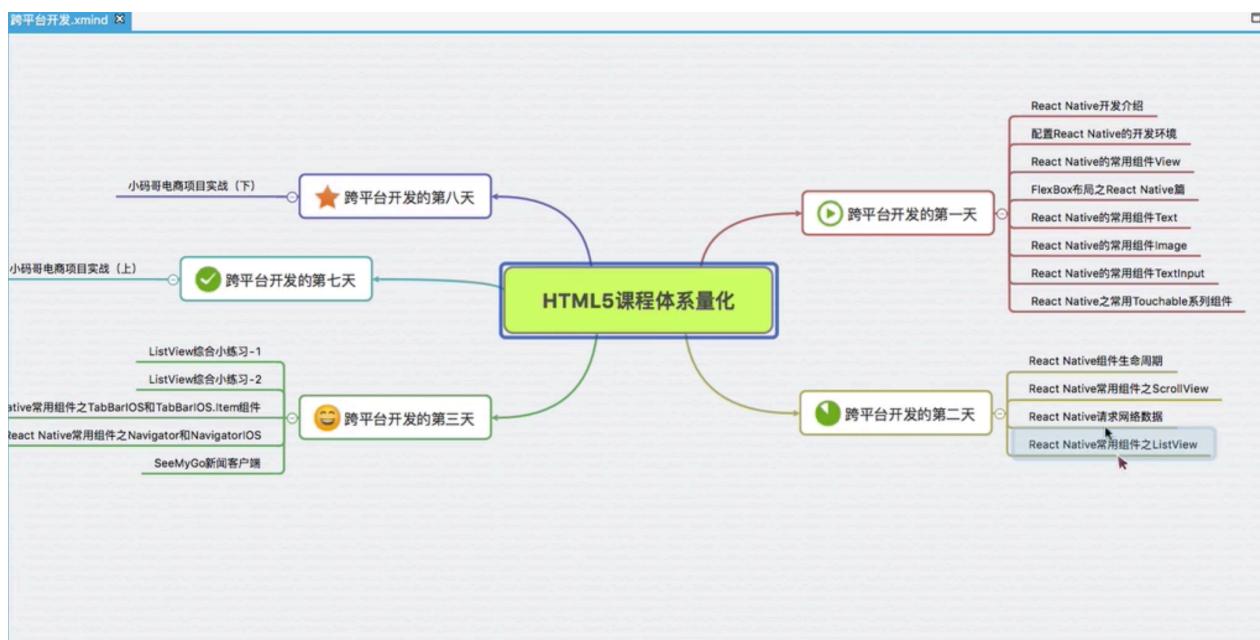
```
<View style={styles.container}>
  <Image source={{uri:'https://www.baidu.com/img/bd_logo1.png'}} style={{flex:1,width:200,
height:100, resizeMode: Image.resizeMode.cover}}/>
  <Image source={{uri:'https://www.baidu.com/img/bd_logo1.png'}} style={{flex:1,width:200,
height:100, resizeMode: Image.resizeMode.contain}}/>
  <Image source={{uri:'https://www.baidu.com/img/bd_logo1.png'}} style={{flex:1,width:200,
height:100, resizeMode: Image.resizeMode.stretch}}/>
</View>
```

细心的同学可能已经注意到，我在上面用到了resizeMode这样一个属性，那么这个属性的作用相当于OC中设置图片的内容模式。

Image.resizeMode.cover: 图片居中显示，没有被拉伸，超出部分被截断；

Image.resizeMode.contain: 容器完全容纳图片，图片等比例进拉伸；

Image.resizeMode.stretch: 图片被拉伸适应容器大小，有可能会发生变形。



二、TextInput的常见属性

因为TextInput是继承自UIView，所以View的属性TextInput也能够使用，一些样式类的属性在学习的时候可以参照View的相关属性。

value 字符串型

文本输入的默认值

onChangeText 函数

监听用户输入的值：

```

render(){
  return(
    <View style={{paddingTop:50}}>
      <TextInput style={testStyle.TextInputStyle}
        onChangeText={({text})=>{this.setState({input: text})}
        placeholder='我是占位文字'
        password={true}
        keyboardType='number-pad'
        multiline ={true}
        secureTextEntry = {false}
      />
      <Text>{` 输入的文字: ' + this.state.input}</Text>
    </View>
  );
}

```

旋之华

keyboardType 键盘类型

决定打开哪种键盘，例如，数字键盘。

```
enum('default', "ascii-capable", 'numbers-and-punctuation', 'url', 'number-pad', 'phone-pad',  
'name-phone-pad', 'email-address', 'decimal-pad', 'twitter', 'web-search', "numeric")
```

multiline 布尔型

如果值为真，文本输入可以输入多行。默认值为假。

password 布尔型

如果值为真，文本输入框就成为一个密码区域。默认值为假。

placeholder 字符串型

在文本输入之前字符串将被呈现出来，通常被称为占位文字

placeholderTextColor 字符串型

占位符字符串的文本颜色

autoCapitalize enum('none', 'sentences', 'words', 'characters')

可以通知文本输入自动利用某些字符。

- characters: 所有字符,
- words: 每一个单词的首字母
- sentences: 每个句子的首字母 (默认情况下)
- none: 不会自动使用任何东西

autoCorrect 布尔型

如果值为假，禁用自动校正。默认值为真。

autoFocus 布尔型

如果值为真，聚焦 componentDidMount 上的文本。默认值为假。

bufferDelay 数值型

这个会帮助避免由于 JS 和原生文本输入之间的竞态条件而丢失字符。默认值应该是没问题的，但是如果你每一个按键都操作的非常缓慢，那么你可能想尝试增加这个。

clearButtonMode enum('never', 'while-editing', 'unless-editing', 'always')

清除按钮出现在文本视图右侧的时机

controlled 布尔型

如果你真想要它表现成一个控制组件，你可以将它的值设置为真，但是按下按键，并且/或者缓慢打字，你可能会看到它闪烁，这取决于你如何处理 `onChange` 事件。

editable 布尔型

如果值为假，文本是不可编辑的。默认值为真。

enablesReturnKeyAutomatically 布尔型

如果值为真，当没有文本的时候键盘是不能返回键值的，当有文本的时候会自动返回。默认值为假。

onBlur 函数

当文本输入是模糊的，调用回调函数

onChange 函数

当文本输入的文本发生变化时，调用回调函数

onEndEditing 函数

onFocus 函数

当输入的文本是聚焦状态时，调用回调函数

returnKeyType enum('default', 'go', 'google', 'join', 'next', 'route', 'search', 'send', 'yahoo', 'done', 'emergency-call')

决定返回键的样式

secureTextEntry 布尔型

如果值为真，文本输入框就会使输入的文本变得模糊，以便于像密码这样敏感的文本保持安全。默认值为假。