# React Native 学习笔记三之 ScrollView,ListView

作者：wyw89500
原文链接：https://ld246.com/article/1483079877920
来源网站：链滴

<h2 id="ScrollView">ScrollView</h2>
<hr>
<ul>
<li>ScrollView 是一个通用的可滚动的容器，你可以在其中放入多个组件和视图，而且这些组件并不要是同类型的。ScrollView 不仅可以垂直滚动，还能水平滚动（通过 horizontal 属性来设置）。</li>

<li>ScrollView 适合用来显示数量不多的滚动元素。放置在 ScollView 中的所有组件都会被渲染，怕有些组件因为内容太长被挤出了屏幕外。如果你需要显示较长的滚动列表，那么应该使用功能差不但性能更好的 ListView 组件。</li>
</ul>
<h2 id="ListView">ListView</h2>
<ul>
<li>ListView 组件用于显示一个垂直的滚动列表，其中的元素之间结构近似而仅数据不同。</li>
<li>ListView 更适于长列表数据，且元素个数可以增删。和 ScrollView 不同的是，ListView 并不立渲染所有元素，而是优先渲染屏幕上可见的元素。</li>
<li>ListView 组件必须的两个属性是 dataSource 和 renderRow。dataSource 是列表的数据源，而enderRow 则逐个解析数据源中的数据，然后返回一个设定好格式的组件来渲染。</li>
<li>下面的例子创建了一个简单的 ListView，并预设了一些模拟数据。首先是初始化 ListView 所需的dataSource，其中的每一项（行）数据之后都在 renderRow 中被渲染成了 Text 组件，最后构成整个ListView。</li>
<li>rowHasChanged 函数也是 ListView 的必需属性。这里我们只是简单的比较两行数据是否是同个数据（===符号只比较基本类型数据的值，和引用类型的地址）来判断某行数据是否变化了。</li>
</ul>
<pre><code class="highlight-chroma">class QRCodePay extends Component{
  // 初始化模拟数据
  constructor(prop
) {
    super(props);
    const ds = new
istView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
    this.state = {
      dataSource: ds
cloneWithRows([
        'John', 'Joel', '
ames', 'Jimmy', 'Jackson', 'Jillian', 'Julie', 'Devin'
      ])
    };
  }
  render() {
    return (
      <View styl
={{flex:1,paddingTop:22}}>
        <ListView
dataSource = {this.state.dataSource}
                rende
Row = {(rowData)=> <Text>{rowData}</Text>}
                style
 {{height:40,backgroundColor:'red',marginTop:5}}
        />
      </View&gt

    );
  }

}
</code></pre>
<h2 id="ListView-">ListView</h2>
<ul>
<li>组件 ListView 组件是 React Native 核心组件之一,应用十分的广泛,主要是高效的展示列表数据</l
>
</ul>
<h2 id="一-基本使用">一.基本使用</h2>
<hr>
<h3 id="1-首先创建一个ListView-DataSource数据源-然后给它传递一个普通的数据数组">1.首先
建一个 ListView.DataSource 数据源,然后给它传递一个普通的数据数组</h3>
<ul>
<li>因为数据源在初始化后会改变,所以放到 getInitialState 方法中,代码如下</li>
</ul>
<h3 id="2-使用数据源-dataSource-实例化一个ListView组件-定义一个renderRow回调函数-这个
数会接受数组中的每个数据作为参数-返回一个可渲染的组件-就是listView每一行的item-">2.使用数
源(dataSource)实例化一个 ListView 组件,定义一个 renderRow 回调函数,这个函数会接受数组中的
个数据作为参数,返回一个可渲染的组件(就是 listView 每一行的 item)</h3>
<pre><code class="highlight-chroma"><span class="highlight
cl">var LVList = React.createClass({

  // 初始化模拟数据
  getInitialState: f
nction() {

    //创建数据源 Da
aSource是大写  返回新数据的条件是当且仅当两行数据不一样的时候返回新数据
    var ds = new Lis
View.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
    this.state = {
        //构造假
据 固定写法,传一个普通的数组
        dataSource
 ds.cloneWithRows(['row 1', 'row 2','row 3']),
      };
  },
  render() {
      return (
        <View st
le={{flex:1}}>
          <ListV
ew
            dataS
urce={this.state.dataSource}
            rende
Row={this.renderRow}
          />
        </View
gt;
      );
    },
    //返回每一行
组件
    renderRow: fun

```
tion(rowData: string, sectionID: number, rowID: number) {
</span></span><span class="highlight-line"><span class="highlight-cl">    return (
</span></span><span class="highlight-line"><span class="highlight-cl">      //添加点击事

</span></span><span class="highlight-line"><span class="highlight-cl">      &lt;TouchablOpacity
</span></span><span class="highlight-line"><span class="highlight-cl">        onPress={(=&gt;{alert('点击了第'+sectionID+'组的第'+rowID+'行')}}
</span></span><span class="highlight-line"><span class="highlight-cl">      &gt;
</span></span><span class="highlight-line"><span class="highlight-cl">        &lt;View style={styles.rowContainer}&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">          &lt;View style={styles.row}&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">            &lt;Image style={styles.thumb} source={{uri:Imgs[0]}} /&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">            &lt;Text style={{flex:1,fontSize:16,color:'blue',marginLeft:50,}}&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">              {roData + '测试~'}
</span></span><span class="highlight-line"><span class="highlight-cl">            &lt;/Text&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">          &lt;/View&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">        &lt;/View&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">      &lt;/TouchableOpacity&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">    );
</span></span><span class="highlight-line"><span class="highlight-cl">},
</span></span><span class="highlight-line"><span class="highlight-cl">})
</span></span></code></pre>
```

## 二.常用属性

- 1.ScrolView 的相关属性样式都全部继承
- 2.`dataSource`:ListViewDataSource 设置 ListView 的数据源
- 3.`initialListSize`:(number) 设置 ListView 组件刚刚加载的时候渲染的列表行数用这个属性确认首页加载的数据数量,而不是花大量的时间渲染大量的数据,提高性能
- 4.`onChangeVisiableRows`: (funtion) (visiableRows,changeRows)=>Voi 当可变的行发生变化时回调该方法.
- 5.`onEndReachedThreshold`:(number) 当偏移量达到设置的临界值时调用 `ononEndReached`
- 6.`ononEndReached`:(function) 当所有的数据行被渲染之后,并且列表往下进滚动,一直滚动到距离底部 `onEndReachedThreshold` 设置的值进行回调该方法,原的滚动时间进行传递(通过参数的形式)
- 7.`pageSize`:(number) 每一次事件的循环渲染的行数
- 8.`removeClippedSubviews`:(bool) 该属性用于提供大数据列表的滚动性能,该用的时候需要给每一行(row)的布局挺添加 over:`hidden` 样式,该属性默认是开启状态
- 9.`renderFooter`:(function)方法 ()=>renderable 在每次渲染过程中头和尾重新进行渲染,如果发现该重新绘制的性能开销比较大的时候可以使用 StaticContainer 容器或者其他适的组件.
- 10,`renderHeader`:(function)方法 (rowData,sectionID,rowID,highlightRow) >renderable 该方法有四个参数,其中分别为数据源中的一条数据,分组 id,行 id,以及标记是否是高选中状态信息

<li>11.<code>renderScrollComponent</code>:(function)方法 (props)=>renderable 该方可以返回一个可以滚动的组件,默认返回一个 ScrollView</li>
<li>12.<code>renderSectionHeader</code>(function)方法 (sectionData,sectionID)=>rendrable 如果设置了该方法,这样会为每一个 section 渲染一个粘性的 header 试图,该试图粘性的效果是刚刚被渲染开始的时候,该试图会处于对应的内容的顶部,然后开始滑动的时候,会跑到屏幕的顶部,知道动到下一个 section 的 header,知道被替换</li>
<li>13.<code>renderSeparator</code> (function)方法 (sectionID,rowID,adjacentRowHighlighed)=>renderable 如果设置了该方法,会在被每一行的下面渲染一个组件作为分割,除了每一个 secton 分组的头部试图前面的最后一行</li>
<li>14,<code>scrollRenderAheadDistance</code>(number) 进行设置当改行进入屏幕多少像素后就开始渲染</li>
</ul>
<h2 id="三-高级特性">三.高级特性</h2>
<ul>
<li>listView 有一些高级特性,包括设置每一组的头部,支持设置列表的 header 以及 footer 试图,当数列表滑动到最底部的时候支持 <code>onEndReached</code> 方法回调,设备屏幕列表可见的试图据发生变化的时候回调 <code>onChangeVisiableRows</code> 以及一些性能方面的优化特性</li

<li>当需要动态加载非常多的数据的时候,可以使用下面的一些性能优化的方法,让滚动更加平滑:
<ul>
<li>1).只更新渲染数据变化的那一行,<code>rowHasChanged</code> 方法会告诉 ListView 组件否需要重新渲染当前那一行</li>
<li>2).选择渲染的频率.默认情况下每一个 event-loop(事件循环)只会渲染一行(可以同 pageSize 自义属性设置),这样可以把大的工作量分割,提高整体的渲染性能.</li>
</ul>
</li>
</ul>
<h2 id="四-注意事项">四.注意事项</h2>
<ul>
<li>在 React Native 中 ScrollView 组件可以使用 <code>stickyHeaderIndices</code> 轻松实现ticky 效果,而是用 ListView 时,使用不生效.</li>
</ul>
<h4 id="如何实现滚动时每个section-header会吸顶呢">如何实现滚动时每个 section header 会吸呢</h4>
<ul>
<li>在 ListView 中要实现 sticky,需要使用 <code>cloneWithRowsAndSections</code> 方法,将code>dataBlob</code>(object),<code>sectionIDs</code>(array),rowIDs(array)三者传进去
<ul>
<li>1>dataBlob 它包含 ListView 所需的所有的数据(section header 和 rows),在 listView 渲染据的时候,使用 <code>getSectionData</code> 和 <code>getRowData</code> 来渲染每一行数据,dataBlob 的 key 值包含 sectionID + rowID</li>
<li>2>sectionIDs:用于标识每组的 seciton</li>
<li>3>rowIDs:用于描述每个 section 里面的每行数据的位置及是否需要渲染,在 ListView 渲染时,先遍历 rowIDs 获取到对应的 dataBlob 数据.</li>
</ul>
</li>
</ul>