征服React Native—列表组件

移动应用往往受限于屏幕大小，而数据内容长度的不确定性，在很多地方都需要用列表组件来作为数据展示的容器。对应于原生应用组件，它可能是iOS的TableView，也可能是Android的ListView，RecycleView。这些组件都有一些共同的特点：

> 视图可滚动。
>
> 可复用视图模板。
>
> 视图高度随着数据内容长度的变化而弹性变化。
>
> 自带性能优化。

当你在纠结要不要用列表组件的时候，可以考虑一下，你使用的场景是否需要具备以上的属性，尤其是性能优化。

在每一列视图的高度都较低的情况下，在手机上一屏的显示内容一般不过7列左右，且不说大部分时候我们的UI不会出这种让人心累的设计，而是我们不能预测出API到底会返回多少组数据回来。所以，掌握列表组件的使用，是作为移动开发必须掌握的一项基本技能。

下面，就让我们来看看React Native的框架之下，又有哪些列表组件可供使用呢。

认识列表组件

ListView

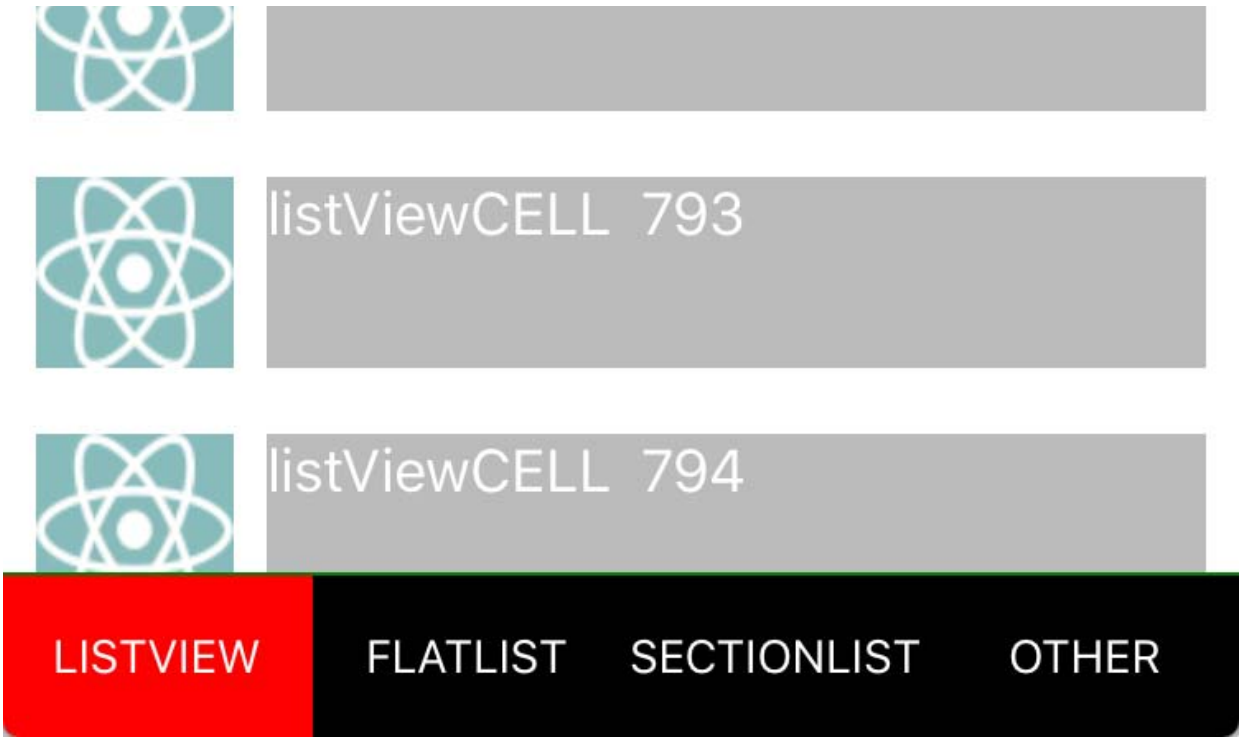
是React Native最早诞生的列表组件，可以方便的用来显示具有纵向滚动属性的数据，实现最基本的两个属性 dataSource 和 renderRow就能让它工作起来。它也支持更多高级的属性，如section和sticky section headers, header，footer，onEndReached等，以及一定的性能优化。为了使ListView滚动更加平滑，在动态的加载一个大的数据（无尽列表）时，可以这样一些优化：

- 只优化发生变化的列：rowHasChanged就是通过比较数据是否发生变化，来判断ListView的row是否需要重绘。
- 限定行渲染的速度：默认每次只渲染一行（可以由pageSize属性来控制），把工作分解为较小的块，以减少渲染时丢帧的几率。

基本用法：

```
<ListView
   dataSource={this.state.dataSource}
   renderRow={(rowData, sectionID, rowID) => this.cell(rowData,
rowID)}
/>
```

但是，ListView在处理无尽列表时，表现却不尽人意，它并不会把视图以外的元素从VirtualDom上面移除，在列表长度长度较大时，滚动时往往出现掉帧情况，内存也占用随着列表的滚动，消耗急剧增加。如上图中所示。



NOTE 从使用者的角度来看，FlatList和SectionList是ListView的一次裂变。不过它们并不是ListView所派生出来，而是同属于VirtualizedList的具体实现，比起ListView，它们在性能上做了极大的改进，最早出现在0.43版本，但在该版本中的bug较多，如果想要使用建议升级RN到0.44以上。

FlatList

顾名思义，它是一个扁平化的列表，砍掉了section的支持，同时，增加了很多移动端常用的玩法：支持横向滑动，下拉刷新，separator，ScrollToIndex等。相比ListView，性能上也得到了巨大的提升，一般情况下，推荐使用FlatList。基本用法：

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

SectionList

如果需要把列表进行分类展示，同时给每个分类设置头部，比如像地址，分类的产品，分类的相册等，SectionList就是最好的选择。

基本用法：

```
<SectionList
  renderItem={({item}) => <ListItem title={item.title} />}
  renderSectionHeader={({section}) => <H1 title={section.key} />}
  sections={[ // homogenous rendering between sections
```

```
      {data: [...], key: ...},
      {data: [...], key: ...}
    ]}
  />
```

VirtualizedList

如果你需要更强的定制化的列表，RN的FlatList和SectionList已经不能满足你要的效果，可以在VirtualizedList上增加Wrapper来实现你的定制化。

虚拟化通过维护有限宽度的渲染窗口，并把渲染窗口之外的所有item替换为空，这样大大提高了大型列表的内存消耗和性能，滚动起来也更加的流畅。

常用属性

data：源数据，默认为Array<{key: string}>类型。

renderItem：对应一个数据Item的显示。

ListHeaderComponent：列表头部。

ListFooterComponent：列表尾部。

ListEmptyComponent：当列表为空的时候，显示的component.

horizontal：是否水平方向展示。

onEndReached：已经滚动到底部的callback.

onRefresh：下拉刷新的callback.

refreshing：标记是否正在刷新。

initialNumToRender：如果有“回到顶部”的需求，建议设置该属性，建议为刚好满屏时候的列数。

Multi-column

如果要实现GridView的效果，你可以FlatList自带的属性numColumns和columnWrapperStyle配合使用，实现多列：

```
<FlatList
  horizontal={this.state.horizontal}
```

```
    data={this.props.data}
    numColumns={2}
    columnWrapperStyle={styles.multiColumns}
    renderItem={({ item, index }) => this.props.renderRow(item,
index)}
  />
```

下拉刷新

ReactNative在新的列表组件当中，已经提供了下拉刷新的功能，可以通过快速的设置refreshing和onRefresh方法来实现：

```
  <FlatList
    data={this.props.data}
    renderItem={({ item, index }) => this.props.renderRow(item,
index)}
    refreshing={this.state.refreshing}
    onRefresh={() => {
      this.setState({refreshing: true})
      this.props.getProducts(this.state.pageIndex)
        .then((items) => {
          this.setState({refreshing: false})
        })
        .catch((error)=> Alert.alert(error.message))
      }
    }
  />
```


超长列表的优化

对于列表的优化，主要集中在两个方面，一个是内存消耗，一个用户响应，用户响应又可以分为：滚动是否流畅，对点击等操作响应速度是否迅速。我们先来看看新的列表组件VirtualizedList都给我们带了哪些改进：

> PureComponent: 减少不必要的渲染，如果props属性不变，它就不会重绘。 这里需要我们确保在更新props后不是===，否则UI可能无法更新更新。
>
> 限定渲染窗口： 通过维护有效项目的有限渲染窗口并把渲染窗口之外的所有元素替换为空（Blank），大大提高了大型列表的内存消耗和性能。
>
> 低优先级渲染窗口以外的区域：窗口适应滚动行为，如果项目远离可见区域，则项目将以低优先级（在任何运行的交互之后）逐渐呈现，否则为了最小化查看空

> 格的可能性。。
>
> 异步渲染：内容将异步地渲染在屏幕外。 这意味着可能滚动会比填充率更快，看到空白的内容。

可以看到，新的列表组件在内存消耗上做出了改进，滚动的流畅度得到了较大的提升，但是，对于用户点击等操作的响应的速度应该算是没有带来利好，反而在滚动中会出现白屏。

从截图中可以看到，FlatList在流畅度的提升还是很明显的，不过，在急速滚动的情况下，中间会出现白屏，这对于用户体验上来说很不友好。这里，我们可以参考VirtualizedList提供的属性来做优化。

windowSize： 限定绘制的最大数目，默认为21。

maxToRenderPerBatch：一次绘制的最大数目。

updateCellsBatchingPeriod：更新绘制的间隔时间。

removeClippedSubviews：移除看不见的subview，目前还有bug，可酌情使用。

initialNumToRender：首次绘制的数目。

getItemLayout：可以用来帮助我们跳过高度和位置的重新运算，当我们的每一个Item高度一致时，设置这个属性可以极大的提高渲染效率。

```
getItemLayout={(data, index) => (
  {length: ITEM_HEIGHT, offset: (ITEM_HEIGHT+ SEPARATOR_HEIGHT) *
index, index}
)}
```

不过尝试了几种以上属性的组合，感觉并不能解决很好的解决白屏问题，这个问题的修复只能期待更新的版本，大家也可以尝试主动提交PR。

以上的操作都是VirtualizedList提供的方法，那对于ListView的卡顿问题，我们也可以模仿FlatList的做法去改进：

> 置空非显示区域的元素：把已经移出屏幕的Item给置空。
>
> 一次加载多个元素：增加一次绘制的元素个数。
>
> 重用列表项：限定只渲染指定的N个item，从N+1之后就重用之前创建的Item。

RN列表组件特别愚蠢的一点是，对于移出界面的节点它并不会去销毁，而是依然保留在Dom结构上，这里，新鲜出炉VirtualizedList也并没有解决问题。

可展开的多级列表

如果只是为了实现类似于Android的ExpandableList的展开收拢效果，而且只有一级子目录的情况下，SectionList就已经可以满足我们的需求，只需要为SectionListHeader绑定onPress事件，在事件中修改展开收拢的状态即可。

如果是多级的列表，比如像这样的地址： 北京.朝阳区.望京街道.XX大厦C栋… 像这种层级比较深的情况下，我们可以用什么方法快速实现呢？这种情况下，建议大家采用FlatList。有兴趣一起探讨的同学，欢迎在6月5日加入我们的在线讨论。

你还需要知道的事

目前的列表组件，不管是Android还是iOS平台，都还是比较顺畅而且使用方便，如果不是对用户体验有着特殊要求，以上的内容应该已经可以满足大部分的使用情况。不过，它们也都有各自的缺陷。在处理超长列表的时候，ListView的短板在于滑动卡顿和内存消耗大，FlatList的问题就在于快速滑动时，可能会看到空白的区域，始终都不尽人意。

除了在RN的组件上做优化，我们还能为我们的列表体验优化做点什么呢？

1. 分页展示：尽量不要一次性加载过多的数据，可以的话，给你的数据加上分页加载，比如一次10条数据，让用户主动去下拉刷新，这个用FlatList来实现还是非常简便的。

2. 使用Native原生组件：需要注意的是，React Native为我们提供了可以跨平台使用的列表组件，它们并不是对应的原生组件，所以它们并不是直接使用到了原生组件的特性，从源码来看，FlatList只是用到原生的ScrollView及其事件，但并没有使用到原生RecycleView带来的好处。所以，如果你的项目对支持超长列表以及用户体验要求高的双重要求，你完全可以选择原生组件，这样，才能更好的利用平台特性。

参考阅读

- 使用Enzyme测试React（Native）组件 | 洞见。
- 前端组件化开发方案及其在React Native中的运用。
- 文中代码示例。