阿东入门系列《四》列表上拉加载下拉刷新

1.列表（本部分内容出入官方文档）

对于这个功能，微信小程序中并没有提供类似于 Android 中 listview 性质的控件，所以我们需要使用 wx:for 控制属性绑定一个数组，用数组中各项的数据重复渲染该组件，来达到列表的效果。

```
<view wx:for="{{array}}">
  {{index}}: {{item.message}}
view>

Page({
  data: {
    array: [{
      message: 'foo',
    }, {
```

```
        message: 'bar'
      }]
    }
})
```

默认数组的当前项的下标变量名默认为 index，数组当前项的变量名默认为 item，当然也可以通过 wx:for-item 和 wx:for-index 指定。

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName.message}}
view>
```

wx:for 也可以嵌套，下边是一个九九乘法表

```
<view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="i">
  <view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
    view>
```

```
  view>
view>
```

block wx:for

类似 `block wx:if`，也可以将 wx:for 用在标签上，以渲染一个包含多节点的结构块。例如：

```
<block wx:for="{{[1, 2, 3]}}">
  <view> {{index}}: view>
  <view> {{item}} view>
block>
```

wx:key

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如  中的输入内容， 的选中状态），需要使用 wx:key 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供

1. 字符串，代表在 for 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。

2. 保留关键字 *this 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数

字，如：

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率。

如不提供 wx:key，会报一个 warning， 如果明确知道该列表是静态，或者不必关注其顺序，可以选择忽略。

示例代码：

```
<switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;"> {{item.id}} switch>
<button bindtap="switch"> Switch button>
<button bindtap="addToFront"> Add to the front button>

<switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;"> {{item}} switch>
<button bindtap="addNumberToFront"> Add to the front button>


Page({
  data: {
    objectArray: [
      {id: 5, unique: 'unique_5'},
      {id: 4, unique: 'unique_4'},
      {id: 3, unique: 'unique_3'},
      {id: 2, unique: 'unique_2'},
```

```
    {id: 1, unique: 'unique_1'},
    {id: 0, unique: 'unique_0'},
  ],
  numberArray: [1, 2, 3, 4]
},
switch: function(e) {
  const length = this.data.objectArray.length
  for (let i = 0; i < length; ++i) {
    const x = Math.floor(Math.random() * length)
    const y = Math.floor(Math.random() * length)
    const temp = this.data.objectArray[x]
    this.data.objectArray[x] = this.data.objectArray[y]
    this.data.objectArray[y] = temp
  }
  this.setData({
    objectArray: this.data.objectArray
  })
},
addToFront: function(e) {
```

```
    const length = this.data.objectArray.length
    this.data.objectArray = [{id: length, unique: 'unique_' + length}].concat(this.data.objectArray)
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addNumberToFront: function(e){
    this.data.numberArray = [ this.data.numberArray.length + 1 ].concat(this.data.numberArray)
    this.setData({
      numberArray: this.data.numberArray
    })
  }
})
```

2.下拉刷新

小程序页面集成了下拉功能，并提供了接口，我们只需要一些配置就可以拿到事件的回调。

1. 需要在 .json 文件中配置。(.json 文件的格式和 app.json 与具体页面的.json 文件的区别，前文已经讲过，有疑问的可以移步。）如果配置在 app.json 文件中，那么整个程序都可以下拉刷新。如果写在具体页面的.json 文件中，那么就是对应的页面，可以下拉刷新。

具体页面的.json 文件：

```
{
    "enablePullDownRefresh": true
}
```

app.json 文件:

```
"window": {
    "enablePullDownRefresh": true
  }
```

2. 在 js 文件中添加回调函数

```
    // 下拉刷新回调接口
    onPullDownRefresh: function () {
        // do somthing
```

```
},
```

3. 添加数据

通常情况下的下拉刷新操作，就是把查询条件重置，让页面显示最新的一页数据。下面是笔者 demo 中的下拉刷新回调接口的代码，同时也是一般情况下的操作流程。

```
// 下拉刷新回调接口
onPullDownRefresh: function () {
    // 我们用 total 和 count 来控制分页，total 代表已请求数据的总数，count 代表每次请求的个数。
    // 刷新时需把 total 重置为 0，代表重新从第一条请求。
    total = 0;
    // this.data.dataArray 是页面中绑定的数据源
    this.data.dataArray = [];
    // 网络请求，重新请求一遍数据
    this.periphery();
    // 小程序提供的 api，通知页面停止下拉刷新效果
    wx.stopPullDownRefresh;
```

```
},
```

3 .上拉加载

同下拉刷新一样，小程序中也提供了用于上拉时回调的接口。官方文档中并没有很详细的介绍，经测试发现，上拉回调的接口并不需要额外的配置（下拉时需要在 .json 文件中配置 `"enablePullDownRefresh": true`），直接在页面滑动到底部时就能拿到回调。

1. 在 js 文件中添加回调函数

```
// 上拉加载回调接口
onReachBottom: function () {
    // 我们用 total 和 count 来控制分页，total 代表已请求数据的总数，count 代表每次请求的个数。
    // 上拉时需把 total 在原来的基础上加上 count，代表从 count 条后的数据开始请求。
    total += count;
    // 网络请求
    this.periphery();
},
```