15 底部滚动加载组件开发

更新时间：2019-08-28 15:33:48

> 什么是路？就是从没路的地方践踏出来的，从只有荆棘的地方开辟出来的。
>
> —— 鲁迅

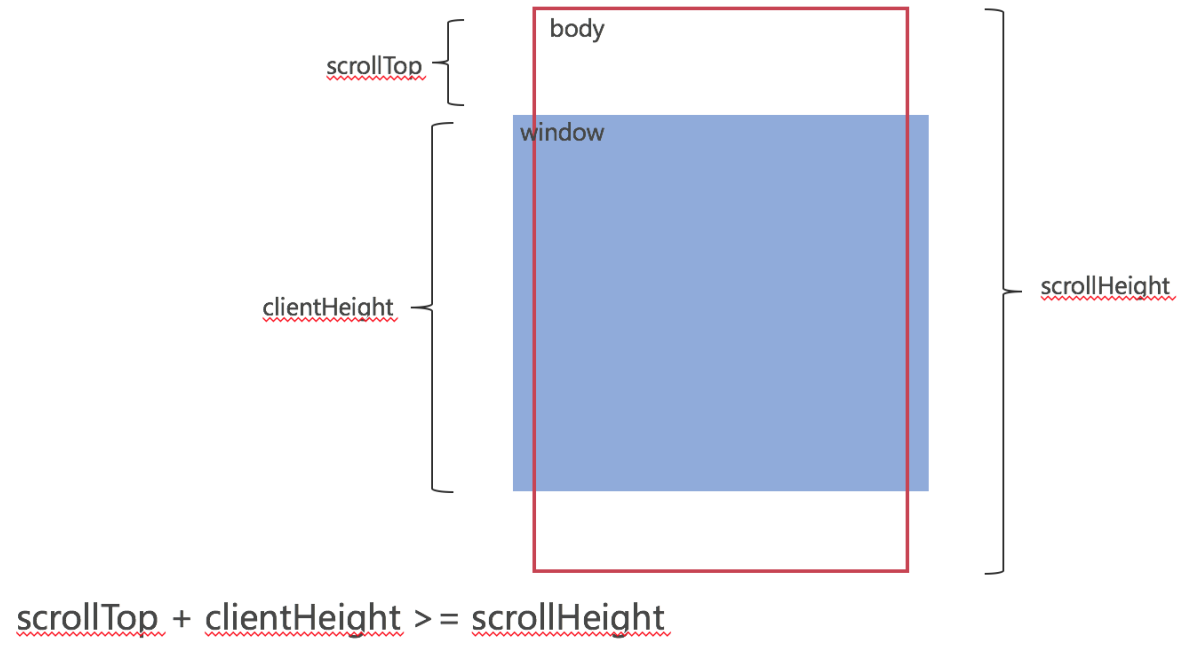
滚动列表组件是移动端项目的一个非常常见也是非常重要的用户交互方式，我们在常见的例如今日头条 APP，美团外卖 APP 等众多的移动端项目中都会用到，同样在我们的朋友圈主页里面，也会用到这种交互方式，那么接下来我们就来实现它。

本章节完整源代码地址，大家可以事先浏览一下：

Github

滚动加载介绍

首先我们来看一张图，给大家讲解一下关于移动端滚动加载逻辑原理，如下图所示：



图上用到的几个概念，我们来解释一下：

scrollTop：表示获取或设置元素的内容向上滚动的像素值。

clientHeight：可见区域的高度，就是如果区域是 `overflow:auto` 的情况下，里面内容如果超过这个高度就会出现滚动条。

scrollHeight：表示获取一个元素的所含内容的高度，包括由于内容超出，滚动出在屏幕上不可见的。

他们之间有着微妙的关系，可以利用这个公式 `(scrollTop + clientHeight) >= scrollHeight` 来实现表示滚动到底部这个点。

同时也可以加一个距离底部还有多少距离的阈值 `proLoadDis`，当 `(scrollTop + clientHeight) >= （scrollHeight - proLoadDis）` 表示距离底部还有 `proLoadDis` 时就提前触发滚动到底部逻辑。

编写滚动组件 ScrollView

在前端项目的 views 文件夹的 wecircle 文件夹下新建 list 页面的 `index.vue`：

```
<div class="list-content" @touchstart="touchmove($event)">
 <scrollView
  @loadCallback="loadCallback"
  :isend="isend"
  :readyToLoad="readyToLoad"
  @scroll="scroll"
 >
  <ul>
   <li v-for="(item,index) in dataList" :key="index">
    <postItem :data="item"></postItem>
   </li>
  </ul>
 </scrollView>
</div>
```

滚动列表的主要内容由 `v-for` 指令来循环渲染出来，`<postItem>` 组件是单个内容组件，我们会在后面的章将来开发。

其中 scrollView 是我们封装的一个滚动加载逻辑组件，我们在前端项目的 components 下新建一个 scrollView 文件夹同时新建组件 index.vue：

```
<div class="scrollview">
  <slot></slot>
  <div class="weui-loadmore" v-show="!isend">
    <i class="weui-loading"></i>
    <span class="weui-loadmore__tips">正在加载</span>
  </div>
  <div class="weui-loadmore weui-loadmore_line weui-loadmore_dot" v-show="isend">
    <span class="weui-loadmore__tips"></span>
  </div>
</div>
```

上面这段代码是滚动组件的主要 UI，isend 这个标志位用来控制是否还可以在继续滚动加载 (当数据没有时，就无法滚动加载了)，同时控制 loading 的显示和隐藏。

下面这段代码主要是监听 onscroll 事件，在滚动时触发，同时获取和计算一些距离值来应用上我们上面的那个公式。

```
mounted () {
  window.addEventListener('scroll', this.onLoadPage.bind(this))
},
beforeDestroy () {
  //在组件销毁时要移除scroll事件
  window.removeEventListener('scroll', this.onLoadPage.bind(this))
},
methods: {
  onLoadPage () {
    //获取clientHeight
    let clientHeight = document.documentElement.clientHeight
    //获取scrollHeight
    let scrollHeight = document.body.scrollHeight
    //获取scrollTop这里注意要兼容一下，某些机型的document.documentElement.scrollTop可能为0
    let scrollTop = document.documentElement.scrollTop || document.body.scrollTop
    //通知父组件触发滚动事件
    this.$emit('scroll', scrollTop)
    //通知距离底部还有多少px的阈值
    let proLoadDis = 30
    //判断是否页面滚动到底部
    if ((scrollTop + clientHeight) >= (scrollHeight - proLoadDis)) {
      //是否已经滚动到最后一页
      if (!this.isend) {

        //判断在一个api请求未完成时不能触发第二次滚动到底部的回调
        if (!this.readyToLoad) {
          return
        }
        //通知父组件触发滚动到底部事件
        this.$emit('loadCallback')
      }
    }
  }
},
```

其中 loading 的样式使用的 weui 的 loading 组件 UI。我们在 vue 的组件里面给 window 绑定事件时，利用 window.addEventListener() 方法，同时需要注意在 mounted 调用在 beforeDestroy 要记得销毁。

slot 插槽

`scrollView` 是一个父组件，在使用时里面的 `<slot></slot>` 将会被替换成 `scrollView` 包裹的子元素的内容，这里的 `<slot>` 叫做插槽：

1. `<slot>` 俗称“占坑”，在组件模板中占好了位置，当使用该组件标签时候，组件标签里面的内容就会自动填坑（替换组件模板中 `<slot>` 位置），当插槽也就是坑 `<slot name="mySlot">` 有命名时，叫做具名插槽，组件标签中使用属性 slot=”mySlot” 的元素就会替换该对应位置内容。
2. 举例说明：

```
<div id="app">
  <children>
    <span>abc</span>
  </children>
</div>
<script>
  var vm = new Vue({
    el: '#app',
    components: {
      children: {
        template:
        "<div id='children'>"+
         "<slot></slot>"+
        "</div>"
      }
    }
  });
</script>
```

这时 #app 渲染出来的内容为：

```
<div id="app">
  <div id='children'>
    <span>abc</span>
  </div>
</div>
```

相当于把 `<slot></slot>` 替换成了 `<span>abc</span>`。

具名插槽：

```
<div id="app">
  <children>
    <span slot="one">abc</span>
    <span slot="two">efg</span>
  </children>
</div>
<script>
  var vm = new Vue({
    el: '#app',
    components: {
      children: {
        template:
        "<div id='children'>"+
         "<slot name="two"></slot>"+
         "<slot name="one"></slot>"+
        "</div>"
      }
    }
  });
</script>
```

这时 #app 渲染出来的内容为：

```
<div id="app">
  <div id='children'>
    <span>efg</span>
    <span>abc</span>
  </div>
</div>
```

OK，理解了插槽的概念，有利于后续的开发，接下来我们要开始调用 API 接口，获取朋友圈的列表数据。

获取朋友圈的列表数据

获取列表数据，我们封装了一个 `fetchData()` 方法，在这个方法里面首先调用 `service.get()` 来调用后台接口获取到数据，然后进入 `formatData()` 对数据进行组装和处理，最后将数据通过 vuex 存入 `store` 的 `setWecircleDataList` 里面。

```
async fetchData () {

  //是否可以发起下一次滚动加载请求的标志位
  this.readyToLoad = false
  //拉取数据
  let resp = await service.get('post/getcirclepost', {
    pageStart: this.pageStart
  })
  //对数据做一下处理
  this.formatData(resp.data)
},
formatData (result) {
  let array = []
  result.forEach(item => {

    array.push({
      id: item._id,//post的id
      avatar: item.user.avatar,//头像url
      nickname: item.user.nickname,//昵称
      content: item.content,//内容
      piclist: item.picList,//图片内容
      comments: item.comments,//评论数据
      time: formatTime(new Date(item.create).getTime() / 1000),//将后台返回的GMT时间转变成本地时间
      isLike: item.isLike,//是否本人已经点赞
      likes: item.likes,//点赞的数据
      user: item.user//发表者的数据
    })
  })
  //通过vuex改变列表的数据
  this.$store.dispatch('setWecircleDataList', array)

  //如果返回的数据为空，证明没有数据了，就把停止滚动加载标志位只为true
  if (result.length === 0) {
    this.isend = true
  }

  //如果第一页并且第一屏的数据太少，就自动再发一次请求，避免内容高度不够，无法触发滚动加载
  if (this.pageStart === 0 && result.length < 4 && !this.lessDataFlag) {
    this.loadCallback()
    this.lessDataFlag = true
  }
  //重制标志位
  this.readyToLoad = true

},
loadCallback () {
  //页数加一
  this.pageStart++
  this.fetchData()
}
```

由上面的代码可以看到，我们的朋友圈列表数据存入了 store 里面，交给了 vuex 来管理，并没有单独的存放在组件的 data 里面原因是：

1. 列表的内容比较复杂，例如列表里的每条朋友圈 post 都可以进行点赞和评论操作，而点赞和评论相关的内容后面都会单独抽离出各自的组件，这就涉及到组件通信。
2. 例如对一个朋友圈 post 点赞后，在不整个列表刷新的情况下，我们需要从 store 里面拿到当前的朋友圈 post，并且找到这个 post 的 like 的数据，然后进行修改，这样可以在不整个刷新的情况下改变一条数据里的一个字段达到更新 UI 的效果。

后面我们在开发单个朋友圈 `<postItem>` 组件时会有针对这里的讲解。

翻页逻辑：

1. 利用 `pageStart` 标志位来存储当前页数，在触发到滚动到底部的方法回调 `loadCallback()` 是会自增。
2. 利用 `readyToLoad` 标志位来保证在前一次的请求没有完成时，不要触发下一次的滚动加载请求。

小结

本章主要讲解了移动 web 滚动加载的原理和朋友圈首页底部滚动组件的相关逻辑开发。
相关知识点：

1. 在实现判断滚动到底部逻辑时，可以利用这个公式 `(scrollTop + clientHeight) >= scrollHeight` 来实现。
2. 在开发 scrollView 组件时，我们在 vue 的组件里面给 window 绑定事件时，利用 `window.addEventListener()` 方法，同时需要注意在 `mounted` 调用在 `beforeDestroy` 要记得销毁。
3. 在开发 scrollView 组件时，`readyToLoad` 标志位很重要，当你滚动的比较快时，会连续触发多次滚动到底部的回调，但是如果利用 `readyToLoad` 标志位就可以加以限制，保证前一次滚动加载逻辑完成后才进行下一次滚动加载的回调。

本章节完整源代码地址：
Github

}

14 顶部组件 UI 开发

16 单条内容组件开发