14 顶部组件 UI 开发

更新时间：2019-09-04 13:59:03

> 世界上最宽阔的是海洋，比海洋更宽阔的是天空，比天空更宽阔的是人的胸怀。
>
> ——雨果

各位小伙伴，项目进行到这里，逐渐进入白热化阶段，接下来我们要开发的页面是整个项目中最复杂的一个页面，好在我们前面已经打下了牢固的基础，如果你前面的技术都掌握了，那么将会很轻易的进行下面的开发，如果还没完全掌握，可以在回顾一下。

本章节完整源代码地址，大家可以事先浏览一下：

Github

朋友圈首页UI效果图

我们先来看看页面的UI效果图，如下所示：

吕小鸣的宝贝1

hi

8天前

♡ 吕小鸣的宝贝

吕小鸣1

你来了

15天前

♡ 吕小鸣,吕小鸣的宝贝

吕小鸣的宝贝1

🐼

15天前

♡ 吕小鸣

吕小鸣1

啦啦操

16天前

♡ 吕小鸣,吕小鸣的宝贝

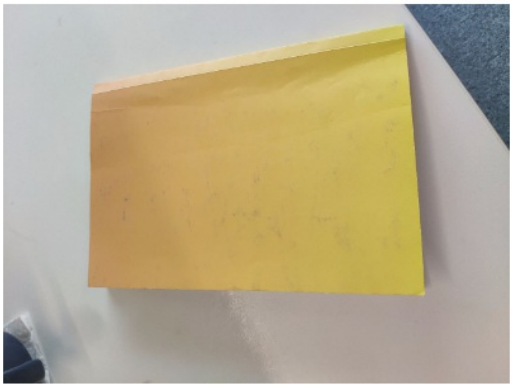
吕小鸣1

鹅鹅鹅才

16天前

董咚咚董1

test post

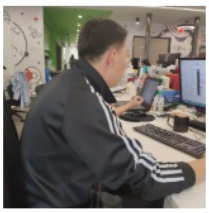


16天前

♡ 董咚咚董

董咚咚董1

工位旁盆景



16天前

♡ 董咚咚董

董咚咚董1

世界如此美好

16天前 ••

董咚咚董:不要如此暴躁

董咚咚董1

啦啦啦啦啦

16天前 ••

♡ 吕小鸣

董咚咚董:你好啊

吕小鸣1

hello

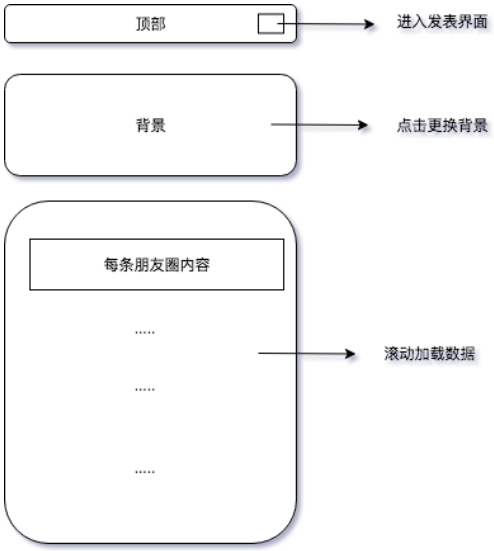
16天前 ••

♡ 董咚咚董

吕小鸣的宝贝:hi

朋友圈首页UI逻辑图

开发页面之前，需要了解一下页面的UI逻辑，如下所示：



整个页面分成3块，我们可以大致拆成3个大组件来写，顶部header组件和背景图片组件和内容列表组件，其中主要的交互逻辑在内容列表这里，接下来，我们就进入开发。

顶部**header**组件逻辑开发

首先我们在前端项目的 views 文件夹下新建一个 wecircle 文件夹 然后在其里面新建 headerbar 文件夹，同时新建 index.vue。

编写 header 组件的template，代码如下：

```
<template>
  <div class="header-bar scale-1px" :class="[headerClass]">
    <p class="title">Wecircle</p>
    <div class="right-icon" @click="goPublish"></div>
  </div>
</template>
```

UI比较简单，需要注意的是我们需要动态设定给 header-bar 的 class 值，因为我们后面有一个动画效果，即默认消失，滚动让离开顶部一定距离后出现，滚动回顶部在消失。

这里我们借助背景颜色来实现这个动画，默认情况下背景颜色是透明表示隐藏消失，给定 `background-color: #ededed;` 表示有背景颜色即出现。动画代码如下：

```
.header-bar {
  height: 57px;
  position: fixed;
  width: 100%;
  transition: all 400ms;
  z-index: 100;
  top: 0;
  left: 0;
  right: 0;
}
.header-bar::after {
  display: none;
}
.header-bar.show::after {
  background-color: #ededed;
  display: block;
}
.header-bar.show {
  background-color: #ededed;

}
```

.show这个 class 就表示显示，代码中给header-bar的伪类after 也应用了样式，是为了让header-bar的底部的一根1px的线，也同步消失和显示。

然后，我们在监听一下滚动事件，控制显示和隐藏，这个事件是由后面章节的滚动加载组件传递而来，代码如下：

```
mounted () {

  this.$bus.$on('showHeader', () => {
    this.headerClass = 'show'
  })

  this.$bus.$on('hideHeader', () => {
    this.headerClass = ''
  })
 },
```

然后，我们给.right-icon绑定一个click事件，来跳转到发表页面，代码如下：

```
  goPublish () {
    // 如果当前用户没有登录，先跳转登录
    if (!this.$store.state.currentUser._id) {
      this.$router.push({
        name: 'login'
      })
      return
    }
    this.$router.push({
      name: 'publish'
    })
  }
```

我们可以通过this.$store.state.currentUser._id判断用户是否有登录状态，否则跳转到登录页面。

顶部背景UI逻辑开发

接下来是页面的顶部背景组件，主要分为背景图片模块和名称头像模块。

首先我们在前端项目的 views 文件夹下新建一个 wecircle 文件夹，然后在其里面新建 index.vue，作为朋友去页面的入口。

背景图片的UI主要是一个div然后设置背景图片来实现，图片的居中显示则利用了CSS的 background-size 属性：

```
background-size: length|percentage|cover|contain
```

length：设置背景图像的高度和宽度。第一个值设置宽度，第二个值设置高度。如果只设置一个值，则第二个值会被设置为 “auto”。

percentage：以父元素的百分比来设置背景图像的宽度和高度。第一个值设置宽度，第二个值设置高度。如果只设置一个值，则第二个值会被设置为 “auto”。

cover：在保持图像的纵横比的前提下，以适合铺满整个容器并将图像缩放成将完全覆盖背景定位区域的最小大小。优点是背景图片全部覆盖所属元素区域；缺点是超出的部分会被隐藏。

contain：与cover相反，在保持图像的纵横比的情况下，以适合铺满整个容器，并将图像缩放成将适合背景定位区域的最大大小。优点是图片不会出现变形，同时背景图片被完全展示出来；缺点是当所属元素的宽高比与背景图片的宽高比不同时，会出现背景留白。

这里我们将会采用 background-size：cover; 比较合适，在保证纵横比的情况下，如果图片超过背景区域，将多余部分隐藏即可，同时设置 background-position: center center; 将主要内容居中显示。

在 index.vue 中，编写下面代码：

```
<div class="top-img" :style="topImgStyle" @click="changeBg"></div>
<div class="name-info">
  <p class="nickname">{{nickname}}</p>
  <img @click="goMyPage" class="avatar" :src="myAvatar">
</div>
```

然后是css样式：

```
.top-img {
  height: 320px;
  width: 100%;
  background-size: cover;
  background-position: center center;
}
```

这里我们绑定了2个事件交互，一个是点击图片来更换背景，一个是点击头像来实现跳转。

下面这段代码主要是获取用户背景图片数据的逻辑，和点击头像跳转的逻辑：

```
computed: {
  topImgStyle () {
    //背景图片首先从store里面获取，获取不到就采用默认的背景图片
    let url =
      this.$store.state.currentUser.bgurl ||
      require('../../assets/topbg.jpg')
    let obj = {
      backgroundImage: 'url(' + url + ')'
    }
    return obj
  }
},
methods:{
    goMyPage () {
    //判断是否登录
    if (!this.$store.state.currentUser._id) {
      this.$router.push({
        path: 'login',
        name: 'login'
      })
      return
    }
    //打开我的页面
    this.$router.push({
      path: 'mypage',
      name: 'mypage',
      params: {}
    })
  },
}
```

goMyPage() 方法是我们点击头像之后的回调，我们可以通过 this.$store.state.currentUser._id 判断用户是否有登录状态，否则跳转到登录页面。

这里将topImgStyle放在 computed 计算属性中就是为了只要 store 里的内容变化，就立刻更新这里的背景图片，例如我们在修改了背景图片之后，这里就会立刻更新。

我们在 changeBg() 方法里去实现更新背景图片:

```
  /*
  * 修改背景图片
  */
  changeBg () {
    //调用weui的actionSheet方法
    weui.actionSheet(
      [
        {
          label: '更换图片',
          onClick: () => {
            this.$refs.uploaderBg.click()
          }
        }
      ],
      [
        {
          label: '取消'
        }
      ]
    )
  },
```

这里使用了 weui 的 actionSheet 组件，可以在这里找到文档。

我们在点击更换图片时，调用了 this.$refs.uploaderBg.click()，模拟 uploader 被点击来进行上传图片，在 template 里已经埋上了一个隐藏 (display:none) 的 uploader，并且在 mounted 的方法里进行了初始化，代码如下：

```
<div style="display:none;" id="uploaderBg">
  <input
    ref="uploaderBg"
    id="uploaderInput"
    class="weui-uploader__input"
    type="file"
    accept="image/*"
  >
</div>
```

下面这段代码是用来初始化我们的 weui 图片上传 uploader 组件，需要注意的是放在 mounted 里是为了保证UI已经渲染完成，这样才能找到相关的 dom 节点进行初始化。

```
mounted () {

  var self = this
  //这里的id需要和dom上一致
  weui.uploader('#uploaderBg', {
    url: service.baseURL + 'post/uploadimg',
    auto: true,
    type: 'file',
    fileVal: 'image',
    compress: {
      width: 1300,
      height: 1300,
      quality: 0.8
    },
    onBeforeQueued: function (files) {
      // `this` 是轮询到的文件, `files` 是所有文件

      if (
        ['image/jpg', 'image/jpeg', 'image/png', 'image/gif'].indexOf(
          this.type
        ) < 0
      ) {
        weui.alert('请上传图片')
        return false // 阻止文件添加
      }
      if (this.size > 10 * 1024 * 1024) {
        weui.alert('请上传不超过10M的图片')
        return false
      }
      if (files.length > 1) {
        // 防止一下子选择过多文件
        weui.alert('最多只能上传1张图片，请重新选择')
        return false
      }

      // return true; // 阻止默认行为，不插入预览图的框架
    },
    onBeforeSend: function (data, headers) {
      // console.log(this, data, headers);
      // $.extend(data, { test: 1 }); // 可以扩展此对象来控制上传参数
      // $.extend(headers, { Origin: 'http://127.0.0.1' }); // 可以扩展此对象来控制上传头部
      headers['wec-access-token'] = service.token
      // return false; // 阻止文件上传
    },
    onSuccess: function (ret) {
      // console.log(this, ret)
      // self.$emit('uploaded',ret)
      // return true; // 阻止默认行为，不使用默认的成功态

      self.submitBg(ret)
    }
  })
},
```

选择完图片之后，会触发 onSuccess 方法，然后调用 submitBg 将新的图片地址存入后台：

```
async submitBg (obj) {
  let resp = await service.post('users/update', {
    userId: this.$store.state.currentUser._id,
    bgurl: obj.data.url
  })

  if (resp.code === 0) {
    //修改成功立刻更新store里的当前user数据
    this.$store.dispatch('setUser', {
      ...this.$store.state.currentUser,
      bgurl: obj.data.url
    })
    weui.toast('修改成功', 3000)
  }
}
```

我们传给后台的实际上是一个url和当前用户的 userid ，在保存成功之后，需要修改成功立刻更新store里的当前user数据，这里需要注意一下。

小结

本章节主要介绍了朋友圈首页的逻辑结构，以及顶部 header 和背景图片模块的开发。
相关技术点：

1. 在使用需要实时更新的UI时，数据最好不要放在`data`里面，而是放在`computed`计算属性中，这样当`store`里的数据有更新之后，UI就会立刻刷新。
2. 在使用weui的uploader组件之后，初始化方法要放在`mounted`里面，这样可以保证UI存在的情况下初始化成功。
3. CSS背景属性`background-size: length|percentage|cover|contain`要理解他们的区别，什么时候用哪个属性要根据UI的情况来决定。

本章节完整源代码地址：
Github

}

13 开发存储帖子接口

15 底部滚动加载组件开发