Vue3Project DAY01

关于vue组件中的响应式变量

1. vue2的做法将`data(){ }`函数return出的所有数据无差别的**都进行数据监听**。一旦在代码执行过程中，将data中的数据进行了改变，则立即触发监听，更新UI。达到数据实时响应的效果。 --- 这种无差别的数据监听很消耗资源。
2. vue3的做法是将数据的自动响应式改为手动，由程序员人为判断哪些变量需要响应式（实时更新）。可以通过代理API多写一些代码来监听变量的变化。
    1. `ref()`代理普通基本数据类型变量
    2. `reactive()`代理复杂对象

Vue数据监听的原理

vue的核心功能就是当管理的数据有变化后，将会及时的更新UI。到底时如何实现的修改变量或对象的属性后，UI可以及时更新?

1. vue2中使用Object.defineProperties()监听对象属性的变化。

   需要对每一个对象、属性都添加监听器，性能堪忧。
2. vue3中使用的时new Proxy()为目标对象创建一个代理对象用于接管所有对目标对象的操作。减少了属性监听器的创建，优化了性能。要求程序员为某些需要响应式的变量手动创建代理。

vue2与vue3的数据响应式源码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vue_data.html 研究v2与v3数据响应式的实现</title>
</head>
<body>
  <h1>vue2数据响应式 Object.defineProperties()</h1>
  <h3 id="num">1</h3>
  <button id="btn1">点击后数字++</button>
  <script>
    // 此处模拟vue2中数据的声明方法，  通过num变量来控制页面中显示的数字
    var data = {
      num: 1
    }
    btn1.onclick = function(){
      data.num++
    }
```

```
  // vue2的做法是监听data兑现， 一旦data中的属性变化，则更新UI
  Object.defineProperties(data, {
    _num: {value: 1, writable: true},
    num: {
      // 当访问data.num时，将执行该get方法，返回_num的值
      get(){
        return this._num
      },
      // 当修改data.num时，将执行set方法，并传入目标值
      set(newValue){
        this._num = newValue
        // 属性修改完毕后，在此处就可以顺便找到页面中用到num的地方
        // 完成DOM更新操作
        let numEle = document.getElementById('num')
        numEle.innerHTML = newValue
      }
    }
  })

</script>

<hr>
<h1>vue3的数据响应式  new Proxy()创建代理</h1>
<h3 id="num2">1</h3>
<button id="btn2">点我数字++</button>
<script>
  // 模拟vue3的data中声明数据
  var v3data = {
    num2 : 1
  }
  // 创建v3data的代理对象 通过new Proxy()代理v3data对象
  var v3dataProxy = new Proxy(v3data, {
    // 当用户设置v3dataProxy的属性时，将会执行该set方法
    // 例如： v3dataProxy.num2=3  时
    set(obj, key, value){
      // 参数1：代理的目标对象
      // 参数2：要设置的属性的属性名
      // 参数3：要设置的属性的属性值
      obj[key] = value        // 先做本职工作，修改属性
      num2.innerHTML = value // 顺便更新DOM
    },
    // 当用户访问代理对象的属性时，执行
    get(obj, key){
      // 参数1：代理的目标对象
      // 参数2：要访问的属性的属性名
      return obj[key]
    }
  })

  btn2.onclick = function(){
    // 操作代理对象，修改代理对象的属性
    v3dataProxy.num2++
  }
```

```
  </script>

</body>
</html>
```

vue3的计算属性

vue2:

```
<span>购物车总价格：{{total}}</span>
```

```
data(){
    return {
        price: 15,
        num: 3
    }
}
computed: {
    total(){
        return this.price*this.num
    }
}
```

vue3:

```
<spam>{{total}}</spam>
```

```
import { ref, computed } from 'vue';
export default defineComponent({
    setup(){
        let price = ref(15);
        let num = ref(11);
        let total = computed(()=>{
            return price.value * num.value
        })
        return {total}
    }
})
```

vue3的监听器

监听器一般用来监听响应变量的变化。一旦响应式变量有变化，则会触发相应监听，执行相关监听方法。

vue2的监听：

```
data(){
    return {
        count: 2
    }
}
watch: {
    count(newval, oldval){
        有变化就执行
    }
}
```

vue3:

```
import { watch } from 'vue';
```

```
// vue3的监听器 watch
watch(count, (newval, oldval)=>{
    console.log(`count变量从 ${oldval} 变成 ${newval}`)
    if(newval==1){
        alert('购买数量不能小于1')
    }
})
```

Vue3.2提供的setup语法糖

在了解了vue3中动态数据绑定的语法、方法声明语法、计算属性语法、监听器语法后，发现几乎所有的代码都在setup里，需要导出的变量写在return后。

3.2提供了setup语法糖来简化语法：

```
<template>
  <div>
    <h3>电影名称: {{ movie.name }}</h3>
    <h3>电影主演: {{ movie.actors }}</h3>
    <hr>
    <button @click="next">点我改名字</button>
  </div>
</template>

<!-- 添加了setup属性的script标签将使用setup语法糖来解析 -->
<script setup lang="ts">
  // 此处编写的代码，相当于在setup方法中编写的代码。并且
  // 在此处声明的变量（例如movie）都会导出，可以在template中使用
  import { reactive } from 'vue'

  const movie = reactive({
    name: '这个杀手不太冷',
    actors: ['lion']
  })
```

```
  function next(){
    movie.name = '交换余生'
    // movie = reactive({
    //   name: '交换余生',
    //   actors: ['大头']
    // })
  }
</script>

<style scoped>

</style>
```

使用Vue框架编程时的代码风格

1. 选项式API （Option API）
2. 组合式API （Composition API）

   组合式API致力于将同一个逻辑所需要用到的变量、方法写在同一个位置，并不是像vue2选项式API一样，写写data、methods、mounted、computed。跳着写。

   组合式更有利于后期查阅代码。

Vue3中通过axios发请求

如果需要在项目中使用axios发送请求，步骤如下：

1. 安装axios

   ```
   npm install axios -S
   ```

2. 在需要的时候，引入`axios`，调用`axios`的方法发送请求

   1. 将MyAxios.js复制过来，改为MyAxios.ts
   2. 通过相应方法，访问相应接口

      ```
      <template>
        <div>
          <h3>电影名称：{{ movie.name }}</h3>
          <h3>电影主演：{{ movie.actors }}</h3>
          <hr>
          <button @click="next">点我改名字</button>
            
          <button @click="listMovies">点我查询电影列表</button>
          <hr>
          <p v-for="item in movieList" :key="item.id">
            电影名称：{{ item.title }}
            上映时间：{{ item.showingon }}
            电影类型：{{ item.type }}
      ```

```
    </p>
  </div>
</template>

<!-- 添加了setup属性的script标签将使用setup语法糖来解析 -->
<script setup lang="ts">
  // 此处编写的代码，相当于在setup方法中编写的代码。并且
  // 在此处声明的变量（例如movie）都会导出，可以在template中使用
  import { reactive, ref } from 'vue'
  import myaxios from '@/http/MyAxios'

  // 声明一个接口Movie，定义电影对象中所有的属性
  import Movie from '@/types/Movie'

  // 处理点击按钮发送电影列表请求相关业务功能
  // const movieList = reactive<Movie[]>([])
  const movieList = ref<Movie[]>([])
  function listMovies (){
    let url = "https://web.codeboy.com/bmdapi/movie-infos"
    let params = {page:1, pagesize:20}
    myaxios.get(url, params).then(res=>{
      console.log('电影列表', res)
      let movies = res.data.data.result // 电影数组
      // movieList.push(...movies)
      movieList.value = movies
    })
  }

  // setup语法糖的测试
  const movie = reactive({
    name: '这个杀手不太冷',
    actors: ['lion']
  })
  function next(){
    movie.name = '交换余生'
    // movie = reactive({
    //   name: '交换余生',
    //   actors: ['大头']
    // })
  }
</script>
```

百慕大影城前台移动端项目实践

项目介绍：该项目供普通用户使用，提供了查询不同类别的电影列表、查看电影详情、查询影院列表、查询电影院中放映厅列表、选择放映厅后选座等功能。

技术选型： Vue3、Typescript、VueRouter、Vuex、Vant组件库。

项目的初始化

1. 新建一个脚手架项目：bmdsutdios-mobile-client

```
# 找一个干净地方：    day01/demo/
vue  create  bmdsutdios-mobile-client
```

依次选择：

```
第一步：
    Manually select features
第二步：
    Babel
    TypeScript
    Router
    Vuex
    CSS Pre-prcessors
    Linter / Formatter
第三步：
    3.x
后续步骤一路回车即可
```

安装模块：

```
cd  bmdstudios-mobile-client
npm  i  axios  -S
```

启动脚手架：

```
npm run serve
```

在项目中引入vant组件库

注意：安装node15以上的稳定版本。自动引入组件的插件需要它。

1. 在项目根目录下安装vant组件库：

```
npm i vant
```

2. 在项目中按需引入组件样式，需要先安装自动引入组件的插件：

```
npm i unplugin-vue-components -D
```

3. 配置VueCLI的配置文件：vue.config.js

```
const { defineConfig } = require('@vue/cli-service')
const { VantResolver } = require('unplugin-vue-components/resolvers');
```

```
const ComponentsPlugin = require('unplugin-vue-components/webpack');

module.exports = defineConfig({
  transpileDependencies: true,
  configureWebpack: {
    plugins: [
      ComponentsPlugin({
        resolvers: [VantResolver()],
      }),
    ],
  },
})
```

搭建项目的初始化布局结构

初始化App.vue的默认结构。

项目分为两大部分：

1. 每个模块的主题内容（上半部分）。
2. 底部选项卡（下半部分）。

实现底部选项卡

详见vant组件库：van-tabbar。