尚硅谷 H5 学科之 mpVue 课程

尚硅谷前端研究院

版本：V 1.0

第一章： mpVue(Vue in Mini Program)

1.1 简介

1. 美团工程师推出的基于 Vue.js 封装的用于开发小程序的框架
2. 融合了原生小程序和 Vue.js 的特点
3. 可完全组件化开发

1.2 特点

1. 组件化开发
2. 完成的 Vue.js 开发体验(前提是熟悉 Vue)
3. 可使用 Vuex 管理状态
4. Webpack 构建项目
5. 最终 H5 转换工具将项目编译成小程序识别的文件

1.3 初始化项目

1. npm install vue-cli -g 下载 vue 脚手架
2. vue init mpvue/mpvue-quickstart my-project 初始化项目
3. cd my-project 进入项目根目录
4. npm install 根据 package.json 安装项目依赖包

5. npm start || npm run dev　启动初始化项目

第二章　　开启项目

2.1　注册小程序

1. src/app.json　　全局配置文件
2. src/App.vue　　等同于原生小程序中的 app.js，可再次写小程序应用实例的声明周期函数 || 全局样式(style 中编写)
3. main.js 应用入口文件，声明组件类型，挂载组件

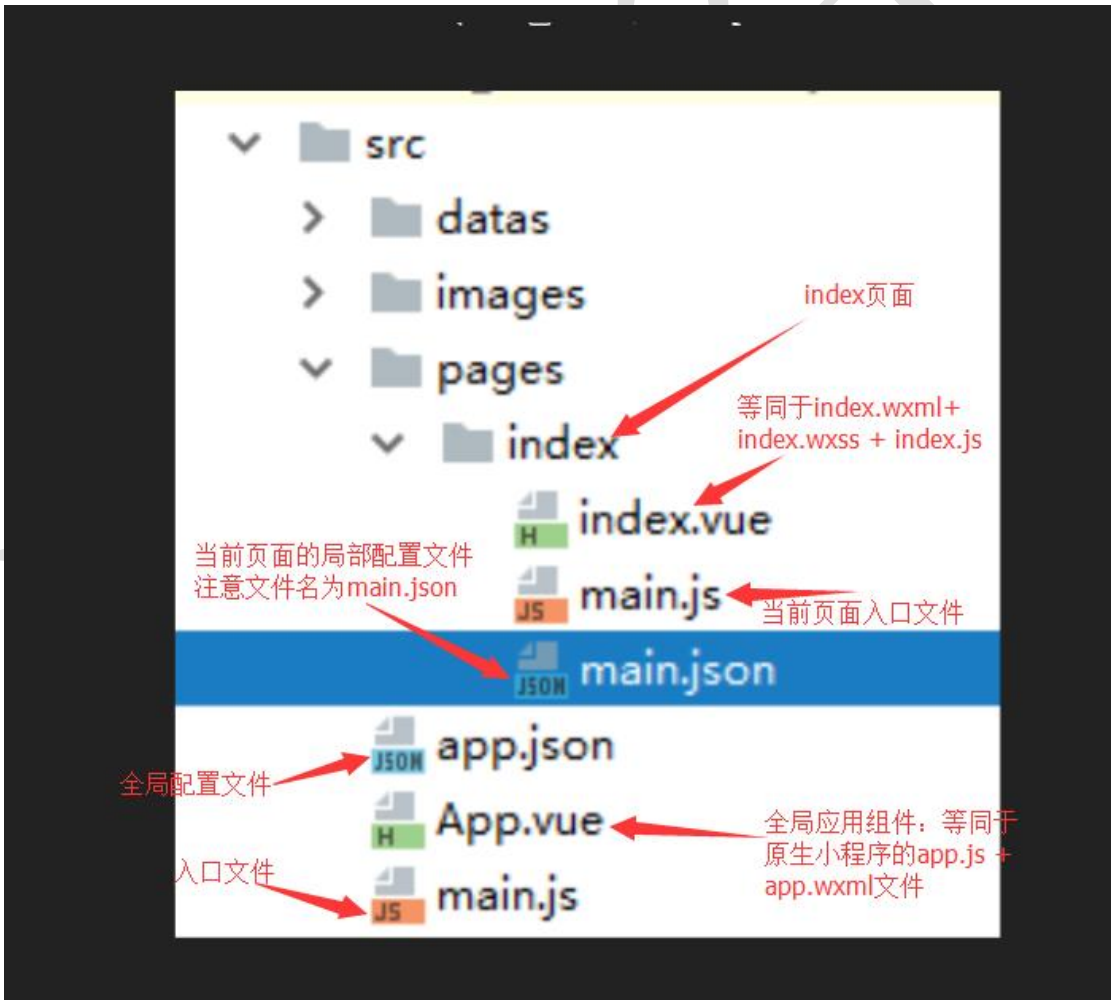
2.2　入口文件介绍

```
import Vue from 'vue'
import App from './App.vue'
// Vue.config.productionTip = false 默认为false，用于启动项目的时候提示信息，设置为false 关闭提示
Vue.config.productionTip = true
// 这个值是为了与后面要讲的小程序页面组件所区分开来，因为小程序页面组件和这个App.vue组件的写法和引入方式是一致的，为了区分两者，需要设置mpType 值
App.mpType = 'app'
// 生成Vue 实例
const app = new Vue(App)
// 挂载组件
app.$mount()
```

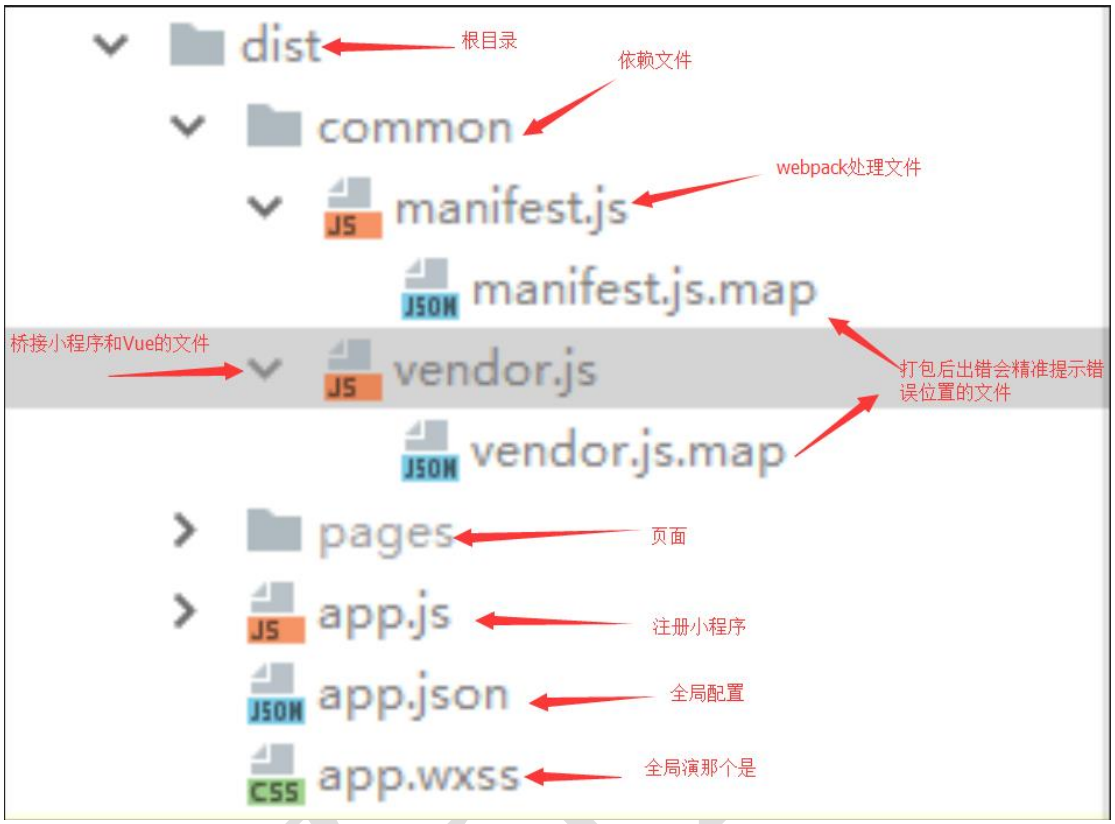
2.3 编写页面 pages/index

2.3.1 页面需要文件介绍

1. index.vue 等同于原生中的 wxml + wxss + js
2. main.js 当前组件页面的入口文件，用于生成当前组件实例，并挂载组件
3. main.json 当前页面的局部配置文件(注意：index.vue 组件最终会被转化为 main.wxml 以及 main.wxss 文件，所以当前的 json 文件需命名 main)
4. src 源文件



5. 自动打包后的 dist 文件



2.3.2 Index.vue 文件

```
<template>
  <div class="indexContainer">
    <img v-show="isShow" :src="userInfo.avatarUrl" alt="">
    <button open-type="getUserInfo" @getuserinfo="handleGetUserInfo"
class="btn" v-show="!isShow">获取用户信息</button>
    <p>hello {{userInfo.nickName}}</p>
    <button @tap="toDetail">开启小程序之旅</button>
  </div>
</template>
```

```
<script>
  export default {
    data(){
      return {
        isShow: false,
        userInfo: {}
      }
    },

    beforeMount(){
      console.log('组件挂载之前')
      // this.getUserInfo()
      this.getUserInfo();


    },
    methods: {
      handleGetUserInfo(msg){
        console.log(this);
        console.log(msg);
        if(msg.mp.detail.rawData){
          this.getUserInfo();
        }
      },
      getUserInfo(){
        // 获取用户信息
       wx.getUserInfo({
```

```
          success: (data) => {
            this.isShow = true;
            this.userInfo = data.userInfo
          },
          fail: () => {

          }
        })
      },
      toDetail(){
        console.log('xxx');
        wx.switchTab({
          url: '/pages/list/main'
        })
      }
    }
  }
</script>

<style>
  page {
    background: gray;
    height:100%;
  }
  .indexContainer {
    display: flex;
```

```
  flex-direction: column;
  align-items: center;
}

.indexContainer img {
  width:200rpx;
  height:200rpx;
  border-radius: 100rpx;
  margin:100rpx 0;
}

.btn {
  width:300rpx;
  height:300rpx;
  border-radius: 150rpx;
  margin:100rpx 0;
  line-height: 300rpx;
}

.indexContainer p {
  font-size: 40rpx;
  font-weight: bold;
  margin:100rpx 0;
}
.indexContainer button {
  background: #009ee0;
  font-size: 32rpx;
```

```
  }
</style>
```

2.3.3 index/main.js

```
import Vue from 'vue'
import Index from './index.vue'
const index = new Vue(Index)
index.$mount()
```

2.3.4 index/main.json

```
{
  "navigationBarTitleText": "主页",
  "navigationBarBackgroundColor": "#8ed145"
}
```

2.4 注意事项

1. 在每个组件中都需要使用： 组件实例.$mount() 去挂载当前组件,否则对应的页面不能生效
2. npm run dev 每次会重新打包 dist 文件，测试只能在小程序工具上
3. mpvue 中绑定小程序原生事件不能使用 bind + 事件名，需要使用@事件名 且要定义在 methods 中否则不生效
4. 新创建的页面需要重新执行: npm run dev 才能将新的页面打包到 dist 文件中

2.5vue 实例声明周期 && 小程序声明周期

2.5.1 vue 实例声明周期

1. beforeCreate 实例初始化之后，数据观测(data observer) 和 event/watcher 事件配置之前被调用。

2. created 实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测(data observer)，属性和方法的运算， watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

3. beforeMount 在挂载开始之前被调用：相关的 render 函数首次被调用。

4. mounted el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。

5. beforeUpdate 数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。 你可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。

6. updated 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

7. beforeDestroy 实例销毁之前调用。在这一步，实例仍然完全可用。

8.destroyed Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。 该钩子在服务器端渲染期间不被调用。

2.5.2 小程序应用 App 实例声明周期

1. onLaunch: 小程序应用初始化
2. onShow: 小程序启动获取后台进入前台
3. onHide: 小程序应用从前台进入后台

2.5.3 小程序页面 Page 实例生命周期

1. onLoad 监听页面加载
2. onShow: 页面显示
3. onReady: 监听页面初始化渲染完成
4. onHide: 监听页面隐藏，注意当前页面实例依然存活
5. onUnload: 监听页面卸载
6. onPullDownRefresh: 监听用户下载动作
7. onReachBottom: 监听用户上拉触底操作
8. onShareAppMessage: 用户点击右上角分享功能
9. onPageScroll: 页面滚动
10. onTabItemTap： 当前是 tab 页时，点击 tab 时触发

2.5.4 注意事项

除了 Vue 本身的生命周期外，mpvue 还兼容了小程序生命周期，这部分生命周期钩子的来源于**微信小程序的 Page**，除特殊情况外，不建议使用小程序的生命周期钩子。

第三章 mpvue 中使用 vue-router && axios

3.1 vue-router

1. 在 mpvue 中对 vue-router 的支持不好,问题较多
2. 进行页面跳转的是可使用小程序提供的 API
   (1) wx.navigateTo() 保留当前页面，可回退
   (2) wx.redirectTo() 不保留，不能回退
   (3) wx.switchTab() 使用于 tabBar 页面

3.2 axios

1. 小程序中不支持使用 axios，会报错： XMLHttpRequest is not a constructor
2. 原因: 小程序的环境和浏览器的环境不一样
3. 解决方法: 使用其他库: flyio

3.3 fly 使用教程

3.3.1 gitHub 地址

https://github.com/wendux/fly

3.3.2 使用步骤

1) 下载: npm install flyio
2) 引入: import Fly from 'flyio/dist/npm/wx' 注意 flyio 支持很多环境下使用
3) 生成实例: let fly = new Fly
4) 配置: Vue.prototype.$fly = fly
5) 使用: 组件中 this.$fly.get()

第四章 mpvue VS 小程序 状态管理

4.1 原生小程序

1) 在 data 中初始化状态数据
2) 修改状态: this.setData({key: value})
3) 页面公共状态：
   a. App 程序实例的 data 中定义
   b. 获取状态数据: let datas = getApp()

c. 修改状态数据: datas.data.xxx = value

4) 或者利用 storage 本地存储

4.2 Mpvue

1) 在组件中通过 getApp 无法拿到对应的数据

2) mpvue 中支持 vuex，所以可以使用 **vuex** 集中管理状态

3) vuex 几个重要的概念:

a. store 对象

b. dispatch() 分发状态

c. actions 携带参与修改状态的数据，并触发 mutations

d. mutations 用于修改状态，并将状态交给 store 对象

e. getter 用于动态计算状态

第五章 原生小程序 VS mpvue 对比总结

1) 原生小程序运行更稳定些，兼容性好，mpvue 可能在某些方面存在兼容性问题（vue-router）

2) mpvue 支持 vue 组件化开发. 效率更高，功能更强大(双向数据绑定, vuex)

3) mpvue 可基于 webpack 组件化, 工程化开发

4) 原生不支持 npm 安装包，不支持 css 预处理

5) 支持 computed 计算属性和 watcher 监听器；模板语法中只支持简单的 js 表达式。可以直接写 div 、span 等标签

computed 的写法

6) 之前会 vue 的工程师上手 mpvue 框架的成本较低