# 前台项目

## 安装需求

- 前端软件需要安装： vscode mysql Navicat for MySQL node.js TS VUE vue-router vuex axios nanoid nprogress qrcode二维码生成插件 vue-lazyload 图片懒加载
  - npm install -g @vue/cli 创建脚手架 vue creata xxx 创建项目
  - vue2 的版本号 vuex 3 vue-router 3 axios 0.24.0 swiper 5 element mysql模块
    - npm i --save less less-loader@5 style 添加 lang (less)
    - vue.directive 全局指令
    - vue.component 全局组件
    - vue.flter 过滤器
- 图片路径问题：遍历图片数组时解决 路径 require("./images/home轮播1.png"),
- filter: grayscale(100%); 灰色遮罩(纪念伟人等)
- 移动css初始化
  - npm install --save xxx
  - npm uninstal xxxx --legacy-peer-deps 卸载
  - import'normalize.css/normalize.css'
- 字体图标使用：https://www.iconfont.cn/
  - 引入css文件 使用 类名 iconfont icon-xxx
  - 引入css文件 设置字体 iconfont 使用 unicode编码
- git解决
  - git push -u origin master 解决error: failed to push some refs to 'github.com:545LHLiang/-.git'==
  - git remote rm origin 解决： fatal: remote origin already exists.
- git 链接gihub ssh使用
  - 生成ssh key ssh-keygen -t rsa -b 4096 -C"2515587601@qq.com" 文件路径c-admin-.ssh-pub
  - ssh -T git@github.com 查看是否配置成功
  - git remote add origin git@github.com:545LHLiang/BISHE23.git
  - git push -u origin master
- public 静态资源(图片) assets 放公用静态资源 components 组件 views路由组件 store vuex配置 router 路由配置
- plugins 自定义插件等 mock 数据 api axios请求 utils 正则 身份验证
- 文件名汇总：search 搜索 xxxNav 导航 Carousel 轮播图 Pagination分页器 router store register注册
- 重点:代理跨域 axios二次封装 路由跳转合并参数 mock vuex 自定义事件(父@xxx=xxx | 子用emit) 全局总线(bus+emit /bus+on) :disabled 禁用按钮

## 滚动行为

- ```
  export default new VueRouter({
      routes,
      // eslint-disable-next-line no-unused-vars
      //滚动行为函数
      scrollBehavior(to, from, savedPosition) {
          return {y:0}  //顶部
      }
  })
  ```

# 项目初始化

- 阿里字体图标：[iconfont-阿里巴巴矢量图标库]
- 使用：生成在线链接 index.html 引入 前面加https:// class='iconfont icon-xxxx
- 浏览器自动打开 "serve": "vue-cli-service serve --open" --open
  - 在vue.config.js中添加
  - devServer: {

    host: 'localhost',

  port: 8080

  }
- eslint校验功能关闭
  - 创建vue.config.js文件：需要对外暴露
  - ```
    module.exports = {
       lintOnSave:false,
    }
    ```
- @配置（vue2自动设置了）
- 代理跨域
- ```
  - devServer: {
    host: 'localhost',
    port: 8080,
    proxy:{
         '/api':{
           target:'http://gmall-h5-api.atguigu.cn', //请求服务器
           //pathRewrites:{'/^api/':''}//路劲重写 如果axios二次封装 设置了 baseURL 则可以不写
         }     }     }
  ```
- 如何网页开发?
  - 写静态-拆分静态组件
  - （二次封装）发请求（mock数据json文件+mockserve.js）
  - -vuex(三联环 dispatch-actions-mutations)
  ```

- 不适用vuex时 开启全局API
- 组件获取仓库数据-动态展示
- ~@ css 中的src 别名
- 如果仓库 服务器无返回值 则 =200成功 失败则返回 Promise.reject(new Error('faile'))
- 正则使用 定义 text=/ / 使用 text.test(验证的变量)

## @常用系统事件

- click 单击 dbclick 双击 mousedown按下 mouseup抬起 mousemove元素中移动 mouseleave离开被选
- mouseout离开被选或者子元素 mouseenter 移入 mouseover 移入被选时
- focus 获取焦点 blur失去焦点 submit 提交表单 keydown 按下 keyup松开
- 键盘事件：keyup.enter/键名称 .delete .tab .esc .space .方位词 .ctrl .alt .shift .meta windows键
- change 改变时 prevent阻止默认 stop阻止冒泡 once只执行一次 passive默认行为立即执行
- input事件 表单文本内容出发变化时触发 无需失去焦点
- $event.target.value 可以获取文本框值

## 引用标签

-
```
<!--长引用-->
<blockquote  cite="http:">
    <p>
        天生我才必有用
    </p>
</blockquote>
<p>
    <!--短引用  cite 表示章节   q表示内容-->  <!--code  表示 代码-->
    在<cite>第一章</cite>中<q>声</q>声明<code>const</code>
</p>
```

# 首页模块

## 路由

### 创建路由

- 创键路由 vue-router 3

```
import Vue from "vue";
import VueRouter from "vue-router";
Vue.use(VueRouter)
const  routes=[]
//path: '/', redirect: '/login' 重定向的路由规则  children:[]子 /不写
//占位符<router-view></router-view>
const router={  routes }

export default router
```

## route和router 区别

- 路由分析：非路由 和 路由
  - 在开发项目时：
    - 书写静态页面(html +css)
    - 拆分组件
    - 获取数据动态展示
    - 完成动态业务逻辑
  - 创建组件时：结构+样式+图片资源
  - 使用非路由组件的步骤（引入 注册 组件标签）

## 路由跳转方式及元信息：

- router-link  to='/'  子路由需加父的path
- 编程式 push|replase可以实现一些业务逻辑
- 组件显示隐藏 使用路由元信息 meta:{k:v} v-show:'$route.meta.k'
  - 有子路由时  写在 孩子里components/component后
- 多个<router-view name="名字"></router-view> name定义名字
  - ==  index.js 中  components:{ path路径，  路由name:组件名，路由name:组件名 }==
- 二级路由可以使用
  - childeren: path 不用反斜杠
  - 路由重定向：redirect：'父组件/子组件'
  - 命名路由：
    - ```
      <router-view />
        <router-view name="helper" />
             components: {
            default: UserProfile,
            helper: UserProfilePreview
          }
      ```

## 传参

- params参数：路径的一部分 需要占位：  冒号占位ke?  （router/index.js）？ 可传可不传
- query参数：不属于路径的一部分 ？ k=v
- 字符串形式
      ```

```
path:'/search/:keyword? ', //params  需要占位  :
this.$router.push('/search/'+this.keyword+"?k="+this.keyword.toUpperCase());
//使用
 我是params参数------{{this.$route.params.keyword}}
 我是query参数------{{this.$route.query.k}}
```

- 模板字符串
- ```
  this.$router.push(`/search/${this.keyword}?k=${this.keyword.toUpperCase()}`)
  ```
- 常用写法

```
//router 中
{
        path:'/search/:keyword',
        component:Search,
        meta: {show:true},
        name:"search"
    },
   //传参
    this.$router.push({
            name:'search',
            params:{
                keyword: this.keyword
            },
            query:{
                k:this.keyword.toUpperCase()
            }
        })
```

- 注意事项:
  - 对象写法式用name和path 都可以 但是path 不可以和params参数同时使用
  - 路由组件参数时params传不传时 在占位符后加? path:'/search/:keyword?'
  - 路由组件传参时 params值为空串时，用undefined解决 params:{ keyword: ''||undefined },
  - 路由组件传递props数据 一般不用
    - 布尔值 prps：true  只支持params参数
    - 对象写法：props：{a=1，b=2}
    - 常用写法：props:($route)=>({keyword:$route.params,k:$route.query})

## 解决编程式路由 抛出异常

- 多次执行相同时 解决方式 重写 push | repalsce
  - 因为最新的vue-router 引入 promise 所以要有成功/失败回调
- 重写 push | repalsce
- ```
  //重写push|repalce
  ```

```
let originpush =VueRouter.prototype.push   //保存一份原有push方法
let originreplace =VueRouter.prototype.replace //保存一份原有replace方法
VueRouter.prototype.push=function(location,resolve,reject){  //重写push  等于一个回调函数（往哪跳，成功回调，失败回调）
    if(resolve && reject)  //如果有成功、失败回调
    {
       // 调用原来的push方法  告诉他往哪跳
        //  originpush()  这样写 this指向 window
        originpush.call(this,location,resolve,reject)  //改变this指向 调用的人    告诉他往哪跳
    }
    else{  //如果没有成功|失败回调  则执行下面语句  添加上回调
        originpush.call(this,()=>{},()=>{})//改变this指向 调用的人    告诉他往哪跳
    }
}
VueRouter.prototype.replace=function(location,resolve,reject){
    if(resolve&&reject)
    {
        originreplace.call(this,location,resolve,reject)
    }
    else{
        originreplace.call(this,()=>{},()=>{})
    }
}
```

# 组件(传参，数据等)

- 全局组件
- 在main.js中
  - import 组件名 from '路径'
  - Vue.component(TopNav.name,TopNav) 注册为全局组件
- 拆分组件时：多个页面有相似结构时 可以拆分成一个全局组件
-

```
<!--banner轮播-->
                    <Carousel :list='bannerlist'></Carousel>
  <!-- 轮播图 -->
                    <Carousel :list="list.carouselList"></Carousel>
```

# 二次封装axios

- 请求拦截器：可以在发请求之前处理一些业务
- 响应拦截器：当服务器数据返回以后，可以处理一些事情
- npm i --save axios
- baseURL：发请求时路径自带路径
- setTimeout：超时多少 请求失败
  - 步骤:引入 变量接收 使用create设置baseURL和setTimeout
  - 请求拦截器 变量.interceptors.request..use(config) 返回config

- 响应拦截器 变量.interceptors.response.use(成功res，失败回调)
  - 成功 res.data 失败 Promise.reject(new Error('faile')

```
//api文件下  request.js文件
//对于axios  进行二次封装
import axios  from "axios"
//利用axios对象方法 create  创建一个axios实例
const requests = axios.create({
    baseURL:'/api',//发请求时路径会出现 api
    setTimeout:5000,//请求时间超过5s

})
//请求拦截器
requests.interceptors.request.use((config)=>{
    //config：配置对象 对象里面含有一个属性很重要  headers请求头
    return config; //直接返回这个对象
})
//响应拦截器
requests.interceptors.response.use((res)=>{
  //成功回调函数：反应数据回来后，
  return  res.data
},(err)=>{
    return Promise.reject(new Error('faile')
})
export default requests
```

## 接口统一管理（解决跨域）

- api/index.js
- 项目小的：在组件的生命周期函数中发请求
- 项目大的：axios.get/post
- 解决跨域：代理跨域
- pathRewrites:{'/^api/':''}//路劲重写 如果axios二次封装 设置了 baseURL 则可以不写
-

```
//vue.config.js中
devServer: {
    proxy:{
      '/api':{
        target:'http://gmall-h5-api.atguigu.cn', //请求服务器
        pathRewrites:{'/^api/':''}//路劲重写
      }
    }
       }
```

## nprogress进度条使用

- npm i nprogress
- 使用

```
//引入进度条
import nprogress from "nprogress";
import 'nprogress/nprogress.css'
//请求拦截器
    nprogress.start()
//响应拦截器
    nprogress.done()
//修改颜色  在 css文件找 .bar
```

## vuex使用

- 创建store文件 并在main.js 进行引入和注册 会多一个 $store属性
- state仓库存储数据
- mutations修改仓库数据唯一手段 commit
- actions 处理action方法 可以书写逻辑 最后提交到mutations  dispatch
- getters 简化仓库数据

### 无小仓库写法

-
```
// dispatch 提交到actions  进行逻辑
this.$store.commit('JIA',1)
this.$store.dispatch('jia',1)
const actions={
   jia(context,vlaue){
      context.commit('JIA',vlaue)
   },
   event(context,vlaue){
    if(context.state.count  %2 ==0){
      context.commit('JIA',vlaue)
}
   }
}
const mutations={
  JIA(state,vlaue){
    state.count += vlaue
  }
}
```

### 带小仓库

- namespaced:true 开启命名空间才可以使用数据
- 每个小仓库格式

-
```
const state={
    b:1
}
const actions={}
const mutations={

    BDD(state){
```

```
        state.b +=1
    }
}
export default{
    namespaced:true,
    state,
    actions,
    mutations
}
```

- 大仓库

```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)
import home from './home'
import sreach from './sreach'
//有小仓库 用module
export default new Vuex.Store({
  modules:{
    home,
    sreach
  }
})
```

- 使用

```
//getters
computed:{
    ...mapState('home',['a']),
     ...mapState ({名字: state=>state.小仓库名字.名字})
   //this.$store.state.home.a
    ...mapState('sreach',['b'])
    //this.$store.state.sreach.b
  },
  methods:{

    ...mapActions('home',['add'])//('home',{方法名:'仓库中的名字'}),
     this.$store.dispatch("home(仓库名)/add",1 (传参) )
    // eslint-disable-next-line no-undef
    ...mapMutations('sreach',['BDD'])//('home',{方法名:'仓库中的名字'})
        this.$store.commit("sreach(仓库名)/add",1  (传参) )
  }
```

## 获取数据

- 在api文件下 创建index.js

- //get 对象形式 多个参数时 url: `/cart/addToCart/${skuId}/${skuNum}`,

```
//所有api接口统一管理  http://gmall-h5-api.atguigu.cn
import requests from "./request";
//服务器   http://gmall-h5-api.atguigu.cn
//三级联动    /api/product/getBaseCategoryList   get
//方法形式: .post  .get
export  const reqCategory =() =>requests.get('/product/getBaseCategoryList')
//对象形式  method: "post",//请求类型  url//请求地址  data//数据
export const reqGetSreachIfo = (params) => requests({
  method: "post",//请求类型
  url: '/list', //请求地址
  data: params,//数据
})
//get 对象形式
export const reqGetDetailInfo = (skuId) => requests({
  method: "get",
  url: `/item/${skuId}`,
})
```

- 在vue文件里 多次请求同一个数据时将他写入app

```
mounted(){
       //通知vuex发请求
  this.$store.dispatch('home/categorylist')  //使用 dispatch 提交到 actions中
}
```

- 在小仓库中

```
import {reqCategory} from '@/api/index'
const state={
    categoryList:[]    //数据类型取决与 服务器返回的数据类型
}
const actions={
    async categorylist({commit}){
        let  result =await reqCategory()
        // console.log(result);
        if(result.code=== 200 ){
            commit('CATEGORYLIST',result.data)
        }
    }
}
const mutations={
    CATEGORYLIST(state,categorylist){
      state.categoryList = categorylist
   }
}
export default{
    namespaced:true,

    state,
```

```
    actions,
    mutations
}
```

## 改变背景颜色

- 列表渲染后 鼠标经过 通过js方式
- 通过定义一个data数据 通过一个方法将列表渲染后的index传入并改变data数据的值
- 动态添加class :class="{cur:currentIndex==index} 如果两个值相等则显示
- 离开时移除 将data数据值改为-1

```
<h3 @mouseenter="changIndex(index)" :class="{cur:currentIndex==index} ">
data() {
    return {
       currentIndex: -1,
    }
  },
  methods:{
    changIndex(index){
        this.currentIndex = index
    }
    outIndex(){
        this.currentIndex = -1
    }
  },
```

## 二三级列表动态显示隐藏

- 在二三级列表父盒子 动态添加  样式 原理同 class  :style="{display:currentIndex==index?'block':'none'}"

## 节流/防抖

- 节流：在规定时间内不会重复执行回调 把频率减少 import throttle from "lodash/throttle" 函数参数(回调函数，毫秒数)
- 防抖：只执行最后一次 improt debounced from lodash/debounced 函数参数(回调函
- 数，毫秒数)

```
//节流    throttle
  <h3 @mouseenter="changIndex(index)" :class="{cur:currentIndex==index} ">
  import throttle from "lodash/throttle" //引入节流函数
  //使用节流函数
  //节流函数名:  throttle(async 写在这)方法
  changIndex:throttle(function(index){
       this.currentIndex = index
   },1000),
```

## 路由跳转传参

- 多级列表跳转时：编程式导航+ 事件委派
- 通过自定义属性：data-xxxx 为a标签添加名字 和id 来区分 一级二级三级标签 通过e.target.dataset事件对象可以获取自定义属性名 如果为标签 则进行路由跳转(名字和几级标签名)

```
//goSearch 最外层父级上添加点击事件
goSearch(e){
        // eslint-disable-next-line no-unused-vars
        //eslint-disable-next-line no-unused-vars
        //点击时获取自定义属性
        let {categoryname,category1id,category2id,category3id} = e.target.dataset  //解构出
        //如果有categoryname 则 进行传参   将名字和 id 传过去
        if(categoryname){
            // let location ={name:'search'}//路由路径
            let query = {categoryName:categoryname}//
            //如果有category1id   以及一级标签
            if(category1id){
                //传参
                query.ccategory1Id = category1id
            }else if(category2id){
                query.ccategory2Id = category2id
            }else{
                query.ccategory3Id = category3id
            }
            // location.query  = query
            this.$router.push({name:'search',query})//挑战到 search组件中 传递query 参数
        }
    }
```

## 不同组件三级列表状态

- 除首页之外的组件 挂载完毕时隐藏三级列表 使用 在三级列表组件的 使用v-show data中添加show数据true
-

```
mounted(){
   //当挂载完毕时，如果不是home组件则隐藏
   if(this.$route.path!='/home'){
       this.show=false
   }
 },
 //鼠标移除时 如果在home组件时   不可以用
 leaveshow(){
      this.currentIndex=-1
      if(this.$route.path!='/home'){
       this.show=false
   }
   }
```

## 过度动画

- <transition name="sort"包裹>过度盒子 </transition

- .名字-enter动画开始　.名字-enter-to 动画结束　 .名字-enter-active  动画样式

```
 .sort-enter{
                height:0
            }
            .sort-enter-to{
                height: 461px;
            }
            .sort-enter-active{
                transition: all .1s linear;
            }
```

## 合并query和params参数

- query参数携带params参数发请求
- params参数携带query参数发请求

```
//query参数携带params参数发请求
this.$router.push({name:'search',params:this.$route.params,query})
//params参数携带query参数发请求
if(this.$route.query){
        let location ={name:'search',params:{ keyword:this.keyword || undefined }};
        location.query=this.$route.query;
        this.$router.push(location);
```

## mock模拟数据

- mock.js生成随机数据，拦截ajax请求 npm i mockjs
- 创建mock文件 常见对应的json文件 代码中注意空格
- 把mock需要的数据放到public文件中 public文件夹打包时候，会原封不动的打包到dist文件中
- 通过mockjs模块模拟数据 mockserve.js文件 （默认对外暴露 图片 json数据格式）
- Mock.mock('请求的地址'，数据)
- main.js引入 mockserve.js
-

```
import Mock from "mockjs"; //引入
import banner from './banner.json'//引入json
import floor from "./banner.json"
Mock.mock('/mock/banner',{code:200,banner})  //请求数据
Mock.mock('/mock/floor',{code:200,floor})
```

## 请求mock数据

- 将二次封装axios复制一个改为 mockxxx  将 baseURL:'/mock',
- index.js 引入mockxxx  获取数据

```
import mockRequest from  "./mockRequest"
export  const resBanner =() => mockRequest.get('/banner')
```

## swiper使用方法

- 安装引包
- 书写html结构
- 实例化对象（vue中 放在）
-

```
import Swiper from 'swiper'
//样式
import 'swiper/css/swiper.css'
//实例化
  var mySwiper = new Swiper ('.swiper', {
    loop: true, // 循环模式选项
    // 如果需要分页器
    pagination: {
      el: '.swiper-pagination',
    },
    // 如果需要前进后退按钮
    navigation: {
      nextEl: '.swiper-button-next',
      prevEl: '.swiper-button-prev',
    },
  })
```

- 注意事项
  - 使用延时器 解决 dispatch中异步语句 导致有v-for的遍历时没有数据
  -

```
setTimeout(()=>{
    // eslint-disable-next-line no-unused-vars, no-undef
   var mySwiper = new Swiper (this.$refs.mySwiper, {
        loop: true, // 循环模式选项
    // 如果需要分页器
    pagination: {
      el: '.swiper-pagination',
    },
    // 如果需要前进后退按钮
    navigation: {
      nextEl: '.swiper-button-next',
      prevEl: '.swiper-button-prev',
    },
  })
    },500)
```

  - 使用watch+$nextTick()解决上面问题
  - immediate:true 立即监听 deep：true 深毒监听

- ==$nextTick(()=>{})：当页面加载完毕后执行==
- 如果请求数据不在本组件中 则可以直接写在mounted中 如需写成watch+$nextTick() 遇到监听不到时开启立即监听
- this.$refs.mySwiper 选择器 slidesPerView:this.skuImageList.length, 显示长度图片长度
- 同时展示多个slidesPerVeiw slidesPerGroup点切换按钮一次几个
- ```
  watch:{
      bannerlist:{
          //immediate:true 当监听不到数据时
          // eslint-disable-next-line no-unused-vars
          handler(newvalue,oldvalue){
          this.$nextTick(()=>{
          // eslint-disable-next-line no-unused-vars
   var mySwiper = new Swiper (this.$refs.mySwiper, {
          loop: true, // 循环模式选项
      // 如果需要分页器
      pagination: {
        el: '.swiper-pagination',
      },
      // 如果需要前进后退按钮
      navigation: {
        nextEl: '.swiper-button-next',
        prevEl: '.swiper-button-prev',
      },
    })
              })
          }
      }
    }
  ```

## 组件传递参数

- 父子组件通信： 父组件用自定义属性 子组件props接受自定义属性名字
- 自定义事件：父：@xxxx='回调'+ 回调方法(数据) 子：$emit('事件名'，数据) 子给父
- 全局事件总线 ：$bus 全能 @on @emit
- 插槽和vuex
- pubsub-js: vue几乎不用

# 搜索模块

## vuex使用：

- post请求需要带参数
- post请求时参数至少是一个空（对象/数组）
  ```

- ```
  export const reqGetSreachIfo = (params) => requests({
    method: "post",//请求类型
    url: '/list', //请求地址
    data: params,//数据
  })
  ```

- 仓库书写是参数默认赋值
- //请求数据 需要多次调用时 把他封装成函数

- ```
  async getsreachlist({ commit }, params = {}) {    // params = {}如果有参数则使用  ，没传参
        let result = await reqGetSreachIfo(params)
        console.log(result);
        // if (result.code === 200) {
        //     commit('GETSREACHLIST', result.data)
        // }
    }
    }
    //请求数据
    this.$store.dispatch('sreach/getsreachlist', {})
    //多次调用时  封装成函数
  mounted() {
   this.getData()
  },
  methods: {
    getData() {
       this.$store.dispatch('sreach/getsreachlist', {})
    }
  }
  ```

## getters使用

- 项目中主要作用为简化仓库中的数据

```
const getters = {
    //state是当前仓库中的state 不是大仓库的state
    attrsList(state) {
        //假如网络不给力或者没有网state.Sreaclist.attrsList应该返回undefined 所以给一个空数据进行遍历
        //state.Sreaclist.attrsLis 返回服务器的数据
        return state.Sreaclist.attrsList||[]
    },
```

# 对象合并

- Object.assign(对象，合并对象)
- Object.assign(this.searchParams,this.$route.query,this.$route.params)
- 扩展运算符...  this.searchParams = {...this.$route.query,...this.$route.params} 数据后期会出bug 慎用

# 重复请求

- 当用户点击三级列表 只能请求一次数据时，可以使用监听route路由发生变化时来实现 多次请求
- 上一次请求没清楚时
-
```
watch: {
  $route() {
    this.searchParams = { ...this.$route.query, ...this.$route.params }
     this.getData()    }
},
```

## 面包屑

- 分类名：删除面包屑时
  - 将参数所对应的数据 清空(参数可有可无时，可以将空字符串写成undefined) 重新发请求
  - this.searchParams.categoryName = undefined this.searchParams.category1Id = undefined
  - 路径改变：
    - 1.路由跳转：自己路由组件跳自己并携带现有params参数
    - this.$router.push({name:'search',params:this.$route.params})
- 处理关键字：
  - 将自己组件的关键字清空 并使用全局事件总线 修改兄弟组件的关键字清空
  -
```
//自己
removeKeyword() {
      this.searchParams.keyword = undefined
      this.$bus.$emit('clear')
    //自己跳自己 并携带现有query参数
      this.$router.push({ name: 'search', query: this.$route.query })
      this.getData()
    }
//兄弟
this.$bus.$on('clear', () => {
            this.keyword=''
         })
```
- 处理品牌：
  - 创建自定义事件 子接受 传递数据  父 回调  操作数据
- 处理售卖参数：
  - 使用自定义事件传给父组件 对应参数  父接受并请求数据
  - 使用v-for 遍历 props数据中的手机属性参数 显示面包屑
  - 数据去重 ： if (this.searchParams.props.indexOf(props) == -1) this.searchParams.props.push(props)

## 自定义事件

- 父使用自定义事件(事件名需要全部小写) 子使用this.$emit 子向父传递数据

- ```
  //父
  <SearchSelector @tradmarkinfo="trademarkInfo"/>
       trademarkInfo(trademark) {
        console.log(trademark);
      }
  //子
  <li v-for="trademark in trademarkList" :key="trademark.tmId"
  @click="trademarkHandler(trademark)">{{trademark.tmName}}</li>
   trademarkHandler(trademark){
        this.$emit('tradmarkinfo', trademark);//自定义事件名,数据
      }
  ```

## 全局事件总线

- ```
  //全局事件总线
   beforeCreate() {
     Vue.prototype.$bus=this
   }
    //通知使用
    this.$bus.$emit('clear')
    //兄弟接受
   mounted() {
         this.$bus.$on('clear', () => {
             this.keyword=''
          })
     },
  ```

## 排序操作

- asc 升序 desc 降序
- 计算属性： isAsc() { return this.searchParams.order.indexOf('asc')!=-1},
- 通过计算属性 判断 isAsc ? '↑' : '↓'
- 点击改变图标
  - ```
    //点击点击事件 @click="changeOrder('1')
    changeOrder(flag) {
         let newOrder
             //flag为标记  记录点击了那个按钮

        if (flag == 1) {
          newOrder=this.searchParams.order.split(':')[1] == 'desc' ? 'asc' : 'desc'
        } else {
          newOrder=this.searchParams.order.split(':')[1] == 'desc' ? 'asc' : 'desc'
        }
        //更新order数据   重新请求数据

        this.searchParams.order = `${flag}:${newOrder}`
    ```
    ```

```
    this.getData()
  }
```

## 分页器

- 需要那些条件?
  - pageNo当前页 pagesize每页多少条，total一共页数 连续页面个数：奇数5或7 continues
  - 对于分页器很重要的地方为 计算出连续页码 起始数字和结束数字
  - totalPage多少页： totalPage() { return Math.ceil( this.total/this.pageSize ); } ceil 向上取整
- 

```
computed: {
   //计算总共多少页  Math.ceil () 向上取整
   totalPage() {
     return Math.ceil( this.total/this.pageSize );
   },
   //计算开始值和结束值
   startNumberendNumberend() {
      //从this中 解构 出 用到的值
     const { continues, pageNo, totalPage } = this
     //定义开始 和 结束 两个变量
     let start = 0
     let end = 0
     //if  连续页的值大于总页数   则开始值为 1  结束值为连续页的值
     if (continues > totalPage) {
       start = 1
       end = continues
     } else {
       //开始值 = 当前页 - (continues / 2)  如:  8 - [ (5 / 2) 2.5取整 2]= 6  从6开始
       start = pageNo - parseInt(continues / 2)
       //结束值 = 当前页 + (continues / 2)   如:  8 + [ (5 / 2) 2.5取整 2]= 10  到10结束
       end = pageNo + parseInt(continues / 2)
       //当 计算出开始值 小于1时，将开始值 为1  结束值 为 连续页数值
       if (start < 1) {
         start = 1
         end=continues
       }
        //当 计算出开始值 大于总页数时，将开始值 为总页数-连续页数的值+1  结束值 为 总页数
       if (end > totalPage) {
          start = totalPage-continues+1
          end =totalPage
       }
     }
     return {start,end}
   }
 }
```

- 分页器动态展示
  - 第一页:当开始值大于1时显示 省略号 大于2时显示
  - 最后一页: 当结束值 小于总页数时显示 省略号 小于 总页数-1时 显示
  - @click="$emit('getpageno',page) 自定义事件 通知父 这是第几页

- 父通过 子使用自定义事件 告知 父当前点击页数 来修改pageNo的值 实现动态

```
//子
<template>
  <div class="pagination">
    <button :disabled="pageNo==1" @click="$emit('getpageno',pageNo-1)">上一页</button>
    <button v-show="startNumberendNumberend.start>1" @click="$emit('getpageno',1)"
>1</button>
    <button  v-show="startNumberendNumberend.start>2" >...</button>
    <!-- 中间部分 -->
    <button v-for="(page,index) in startNumberendNumberend.end"  :key="index" v-
show="page >= startNumberendNumberend.start" @click="$emit('getpageno',page)">{{page}}
</button>
      <button v-show="startNumberendNumberend.end<totalPage-1">...</button>
    <button v-show="startNumberendNumberend.end<totalPage"
@click="$emit('getpageno',totalPage)" >{{ totalPage }}</button>
    <button :disabled="pageNo==totalPage" @click="$emit('getpageno',pageNo+1)">下一页
</button>
    <button style="margin-left: 30px">共{{ total }}条</button>
  </div>
</template>
//父
 getpageno(pageNo) {
      this.searchParams.pageNo = pageNo
       this.getData()
     },
```

## 详情页

- 路由跳转时
  - 带参数 <router-link :to="`/detail/${good.id}`"><img :src="good.defaultImg"  /></router-link>
  - 路由设置：  path: '/detail/:skuid?',
- 放大镜
  - 放大镜布局
    - 原理

```
<div class="spec-preview">
    //图片
   <img :src="imgObj.imgUrl"/>
       //放大镜事件
   <div class="event"></div>
    //大图
   <div class="big">
     <img :src="imgObj.imgUrl" />
   </div>
    //遮罩
   <div class="mask"></div>
 </div>
```

- 放大镜底部轮播图 slidesPerView:数字或小数 显示个数 slidesPerGroup：:数字或小数 切换个数

```
watch: {
  skuImageList:{
    handler() {
      this.$nextTick(() => {
        // eslint-disable-next-line no-unused-vars
        var mySwiper = new Swiper(this.$refs.mySwiper, {
          slidesPerView: 3,//this.skuImageList.length,
          slidesPerGroup:1,
  // 如果需要前进后退按钮
  navigation: {
    nextEl: '.swiper-button-next',
    prevEl: '.swiper-button-prev',
  },
})
      })
    }
  }
}
```

  - 点击添加高亮 自定义一个data数据 动态添加类名 当data数据等于index 时 添加 点击事件传入index 将data数据=index

  - ```
    //
    data() {
        return {
          currentIndex:0
        }
      },
           chang(index) {
          this.currentIndex = index
           }
    ```

- 点击底部轮播 切换图片和放大镜图片
  - 通过全局事件总线 传过索引值 进行修改src

  - ```
    //底部轮播图
    chang(index) {
        this.currentIndex = index
        //通知兄弟组件
        this.$bus.$emit('chang',index)
      }
      //放大镜组件
    data() {
      return {
        index:0
    ```

```
  }
}  ,
  mounted() {
  this.$bus.$on('chang', (index) => {
      this.index = index
  })
},

```

- 放大镜 移动
  - 遮罩移动
    - 遮罩鼠标居中: 获取鼠标位于x 和 y的位置 left = 鼠标位置-元素宽度的一半  top 同理
    - 遮罩跟随鼠标:  将算出来的值赋值给元素+px
    - 约束范围   小于0 时  left  =0    大于元素宽度时  left等于元素宽度  top 同理
    - 大图 显示的left位置  因为 大图为原图的二倍  所以  图片显示负2倍的left/top的位置
  -

```
<div class="event" @mousemove="hanler" ></div>
hanler(event) {
      let mask = this.$refs.mask //获取元素
       let  big = this.$refs.big
      //获取left    鼠标移动距离x的距离 减去   遮罩元素本身宽度的一半
      let left = event.offsetX - mask.offsetWidth / 2
      let top = event.offsetY - mask.offsetHeight / 2
      //约束范围
       if (left < 0) left = 0
     //小于0 时   left   =0      大于元素宽度时   left等于元素宽度
      if (left > mask.offsetWidth) left = mask.offsetWidth
       if (top < 0) top = 0
       if (top >mask.offsetHeight) left = mask.offsetHeight
      //修改元素位置 实现 鼠标跟随效果
      mask.style.left = left + 'px'
      mask.style.top = top + 'px'
       big.style.left = - 2 * left + 'px'
      big.style.top= - 2*top + 'px'
    }
```

- 型号点击高亮
  - 添加高亮类名：通过服务器返回的数据 如果 ischecked为1 则添加
  - 动态：通过foreach 遍历服务器返回的属性数据 找到 ischecked 将全部改为 0 在将自己变为1
  - foreach遍历对象 对象.foreach(箭头函数(函数执行语句))

```
//c2 v-for 遍历属性数组
:class="{ active :c2.isChecked ==1}"
@click="changeActive(c2,color.spuSaleAttrValueList)"
changeActive(saleAttrValue,arr) {
      // console.log(isChecked);
      arr.forEach(item => {
        item.isChecked = '0'
      });
      saleAttrValue.isChecked='1'
    }
```

# 购物车

## 加入购物车

- 用户点击数量输入实现：双向绑定数据 点击时传入数据 并将其修改
  - 处理用户输入非数字情况:
    - 正则： let num = /^\d+$/　　判断他真假 num.test(value)　假为1 真　修改值
    - 文本 * 1　：当用户输入非数字时 乘1 会变成NaN　通过isNaN()方法判断 假时 修改　真 为1

- 按钮实现：
  - 1、发请求将产品加入数据库 (携带产品id 和 数量)
  - 2、服务器存储成功。
  - 3 、失败，给用户提示
  - 4、告诉服务器你是谁
  - 通过请求头 传入你的 token/uuid(使用见UUid) 仓库中state存储： uuid_token:getUUID(),
  - async 和await 成对出现
  - Promise.reject(new Error(result.message)) 失败返回

```
    //h获取购物车
        async getAddcartinfo({ skuId, skuNum }) {
            //服务器写入成功 返回 200
            //因为服务器没有返回其余数据，因此不需要三连环
            let result = await reqAddOrUpdateShopCart(skuId,skuNum)
            //成功
               if (result.code === 200) {
                return '成功'
                   //失败
            } else {
                return Promise.reject(new Error('faile'))
            }
        }
async  addShopcar() {
    try{
        //成功


        await  this.$store.dispatch('detail/getAddcartinfo',
```

```
{skuId:this.$route.params.skuId,skuNum:this.skuNum})
    }catch(error){
        //失败
        alert(error.message)
    }
    },

```

## 本地存储实现加入购物车跳转传参

- 本地存储：持久化的 ---------5M localStorage
- 会话存储：非持久(浏览器关闭则失效) sessionStorage
  - 注意: 一般存储的是字符串 路由传参 产品数据比较复杂的存储是通过会话存储
  - 存储对象时: 使用JSON.stringify()方法 转换为字符串 parse方法 转换为对象
-

```
//存储
sessionStorage.setItem('SKUINFO',JSON.Stringify(this.skuInfo))
//使用
JSON.parse(sessionStorage.getItem('SKUINFO'))
```

## 展示购物车

- 根据身份展示购物测
  - token/ uuid 生成唯一标识

## uuid

- 新建一个utils文件夹 放一些常用的功能模块（正则...临时身份）
-

```
//uuid.js
import { v4 as uuidv4 } from 'uuid'
//生成随机字符串  持久存储
export const getUUID = () => {
    //如果本地存储有uuid则 返回  没有 则获取 并返回
    let uuid_token = localStorage.getItem('uuid')
    if (!uuid_token) {
        uuid_token = uuidv4()
        localStorage.setItem('uuid',uuid_token)
    }
    return  uuid_token
}
//仓库中获取uuid_token
import { getUUID} from '@/utils/uuid_token'
       state: {
        uuid_token:getUUID(),
    },
```

- 在请求拦截器引入store 并在请求拦截器中

- ```
  //请求拦截器
  requests.interceptors.request.use((config)=>{
      //config: 配置对象 对象里面含有一个属性很重要  headers请求头
    //如果仓库中有token或者uuid时
    if (store.state.detail.uuid_token) {
      //请求头添加字段   和后台老师商量好  一般为userTempId
      config.headers.userTempId = store.state.detail.uuid_token
    }
      nprogress.start()
      return config; //直接返回这个对象
  }
  ```

- 购物车 计算总价
  - 使用forEach遍历数组( sum+=数量*价格*)
  - ```
    totalPrice(){
        let sum = 0;
        //购物车商品数组
        this.cartInfoList.forEach(item => {
           sum += item.skuPrice * item.skuNum
        });
        return sum
      }
    ```

- 数组 every方法
  - 遍历数组中每个一个元素 如果全部为 你想要的 则为true 否则 为 false
  - ```
    let arr = [{cur:1},{cur:2},{cur:3}]
    let result = arr.ervey(item=>item.cur==1)   // false
    let arr = [{cur:1},{cur:1},{cur:1}]
    let result = arr.ervey(item=>item.cur==1)   // true
    ```

- 使用 every 判断是否全选
  - ```
    isAllCheck() {
         return this.cartInfoList.every(item=>item.isChecked==1)
       }
    ```

- 购物车 数量需要发请求
  - ```
    async handler(type,disNum,car){
        //type  标记谁出发的
        //disNum 变化量(+1)/(-1)
        //car 为传递商品信息

        switch (type) {
          case 'add': disNum = 1
    ```
    ```

```
                break
        case 'minus':  disNum =  car.skuNum > 1  ?  -1 : 0
                break
        case 'change':
         //如果输入的是非数字  则给服务器传递0
         //是整数或者小数    返回整数 减去服务器的数量
            if(isNaN(disNum) || disNum <1){
               disNum =0
            } else {
              disNum=parseInt(disNum) -car.skuNum
            }
               break
      }
     // 捕获 服务器返回的信息  成功则 刷新页面   失败 打印错误
      try {
          //派发 请求  传入 id 和   增加或减少的数量
        await this.$store.dispatch('detail/getAddcartinfo', { skuId: car.skuId,
skuNum: disNum })
        this.getDate()
      } catch (error) {
        console.log(error.message);
      }

    }
```

- 购物车删除操作
  - 需要删除的商品的id
  - ```
    //vue
     async  deletecart(car) {
        try {
          await this.$store.dispatch('shopcart/deleteshopcartinfo', car.skuId)
          this.getDate()
        } catch (error) {
          console.log(error.message);
        }
        }
      }
      //仓库
        async deleteshopcartinfo({ commit }, skuId) {
             let result = await deletShopCart(skuId)
             if (result.code == 200) {
                 return '成功加入购物车'
             } else {
                 return Promise.reject(new Error('faile'))
             }
         }
    ```

- 切换商品选中状态
    ```

```
//切换商品
    //传入两个值 一个id 一个状态  通过 event 事件 获取
   async changeIschecked(car, event) {
     let isChecked = event.target.checked ? 1:0
     try {
       await this.$store.dispatch('shopcart/checkshopcart',
{skuId:car.skuId,isChecked})
       this.getDate()
     } catch (error) {
       console.log(error.message);
      }
   }
```

```
```

- 删除选中的全部产品
  - 注意 没有一次删除很多产品的接口。可以通过 多次执行一个代码 promise
    - promise.all([p1,p2,p3]) p1-3: 每一个都是promise对象，如果有一个失败则都失败。
    - 通过 dispatch 调用 dispatch 通过遍历 购物车产品数量来 获取 被选中的状态和id
    -
```
    deleteAllCheckedCart({dispatch,getters}) {
        //context:小仓库【提交mutations 、getters、dispatch、state
        let PromiseAll=[]
        getters.shoplist.cartInfoList.forEach(item=> {
            let Promise = item.isChecked == 1 ? dispatch('deleteshopcartinfo',
item.skuId) : ''
            PromiseAll.push(Promise)
        });
        //只要全部成功则成功 一个失败则失败
        return Promise.all(PromiseAll)
    }
```

- 全部产品状态的勾选
  - 当无产品 时 解决全选按钮勾选状态 数组长度大于0时且 所有商品都勾选
  - 传入 全选按钮勾选状态的值 派发修改dispatch
  -
```
//全选按钮控制
   async updateChecked(event) {
     let isChecked=event.target.checked ? '1':'0'
     try {
       await this.$store.dispatch('shopcart/updateChecked',isChecked)
       this.getDate()
     } catch (error) {
       console.log(error.message);
     }
   }
   updateChecked({ dispatch, getters }, isChecked) {
```

```
            let PromiseAll=[]
            getters.shoplist.cartInfoList.forEach(item => {
               let Promise= dispatch('checkshopcart', { skuId: item.skuId, isChecked })
                PromiseAll.push(Promise)
            })
            return Promise.all(PromiseAll)
        }
```

# 登录注册

- 登录注册和git必须会

## 注册

- 书写 双向绑定data数据
- 获取虚假验证码 定义正则 判断手机号是否正确 如果正确则 服务器返回的数据 显示出来

```
async getCode() {
      try {
         let phone = /^(?:(?:\+|00)86)?1\d{10}$/
        let iscode = phone.test(this.phone)
        if (iscode) {
          await this.$store.dispatch('user/getcodeinfo', this.phone)
             this.code = this.$store.state.user.code
        }
      } catch (error) {
        console.log(error.massage)
      }
    }
//仓库
  async getcodeinfo({ commit }, phone) {
            let result = await reqCode(phone)
            console.log(result);
            if (result.code == 200) {
                commit('GETCODEINFO',result.data)
            } else {
                return Promise.reject(new Error('失败 '))
            }
        },
```

- 注册

```
//注册
   async Register() {
     try {
       const {phone,code,password,dbpaswd }= this
      phone&&code&&password==dbpaswd &&await this.$store.dispatch('user/getRegister', {phone,code,password})
          this.$router.push('/login')
```

```
    } catch (error) {
      console.log(error.message);
    }
  }
 //仓库
async getRegister({ commit }, data) {
          let result = await reqRegister(data)
          if (result.code == 200) {
              alert('注册成功')
          } else {
              return Promise.reject(new Error('失败'))
           }
      }
```

- 表单验证 等

## 登录

- 1、获取token 并且 账户密码正确
- 2、带token登录 请求数据

```
//请求拦截器 携带 token   请求数据
//如果用户仓库中 token 存在  则携带
//二次封装
  if (store.state.user.token) {
    config.headers.token = store.state.user.token
  }

```

- 持久化 存储token 本地存储
  - 将token 初始化为 本地获取token 未登录 进去 token之时 null 登陆后 保存token 则变为真实token
  - ```
    //仓库中 state 代码
      token: localStorage.getItem('token'),
    //dispatch
            async getlogin({commit},data) {
                let result = await regLogin(data)
                // console.log(result);
                if (result.code == 200) {
                    commit('GETLOGIN', result.data.token)
                    //本地存储
                    localStorage.setItem('token', result.data.token)
                    return 'ok'
                } else {
                    return Promise.reject(new Error('失败'))
                }
            },
    ```
  - 目前存在问题 (导航守卫)
    ```

    - 进行路由跳转  用户信息必须有  其他组件刷新  用户信息丢失
    - 用户已经登录  还能回去登录
  - 登录
  - 
```
async login() {
     //整理参数
     const { phone, password } = this;
     //在发登录请求
     try {
       //登录成功
       await this.$store.dispatch("userLogin", { phone, password });
       let goPath = this.$route.query.redirect||'/home';
       //跳转到首页
       this.$router.push(goPath);
     } catch (error) {
       alert(error.message);
     }
   },独享守卫
```

## 退出登录

- 
```
   //退出登录
      async getlogout({ state}) {
          let result = await logout()
          if (result.code == 200) {
              state.token = ''
              localStorage.removeItem('token')
              state.userinfo = {}
              return 'ok'
          } else {
              return Promise.reject(new Error('退出失败'))
          }
      }
  //方法
  async logout() {
       //发请求 服务器清楚数据   本地存储等用户信息清楚
     try {
         await this.$store.dispatch('user/getlogout')
         this.$router.push('/home')
     } catch (error) {
         return error.message

     }
   }
```

# 导航守卫

- 导航：路由发生改变

- 守卫：安检员 检查能否进站
- 全局守卫：火车进站安检员
- 路由独享守卫： 检票员
- 组件内守卫：火车查票员

## 全局守卫

- 写在router.js中
- 项目中只要发生路有变化 就可以检测到 分为前置 解析 后置守卫
  - 未登录之前想去的地 登录之后 跳过去 使用query参数 next('/login?redirect='+toPath);

```
router.beforeEach((to, from, next)=> {
    //to : 跳转到那里
    //from : 从哪里来
    //next: 是否可以去    使用  next()     next('/login') 放行指定路由    next(false) 稍后讲
})
```

-

```
router.beforeEach(async (to, from, next) => {
    // next()
    //已经登录  无法去登陆
    let token = stroe.state.user.token
    //多个组件公用的数据
    let name=stroe.state.user.userinfo.name
    if (token) {
        //用户已登录 还想去login  不能去  停留在首页
        if (to.path == '/login') {
            next('/home')
            alert('已登录,无需再次登录')
        } else {
           //如果用户信息有 则放行
            if (name) {
                next()
            } else {
                //如果没有用户信息，则派发
                try {
               await stroe.dispatch('user/getuserinfo')
                  next()
              } catch (error) {
                //token 失效
                    //清楚token    调用退出的请求
                  await  store.dispatch('user/getlogout')
                  next('/login')
              }
            }
        }
    } else {
    //用户未登录||目前的判断都是放行.将来这里会'回手掏'增加一些判断
        //用户未登录:不能进入/trade、/pay、/paysuccess、/center、/center/myorder
/center/teamorder

        let toPath = to.path;
```

```
        //当路径中有trade字段 或者 pay center 时
        if (toPath.indexOf('trade') != -1 || toPath.indexOf('pay') != -1 ||
toPath.indexOf('center') != -1) {
          //未登录之前想去的地  登录之后  跳过去 使用query参数
            next('/login?redirect='+toPath);
        } else {
            next();
        }
    }
})
```

## 路由独享守卫

- 只负责这条路 写在 routers {}中
-

```
{
    path: '/pay',
    component: pay
    beforeEnter:{to,from,next}=>{
        if(from.path=='/trade'){
            next()
        }else{
            next(false)
        }
    }
}

```

## 组件内守卫

- 写在组件内 不常用
    -

```
beforeRouteEnter(to,from,next){
}
beforeRouteupdate(to,from,next){
    动态参数的路径     /foo/1    /foo/2 时使用
}
beforeRouteLeave(to,from,next){

}
```

# 结算页面

- 动态展示服务器数据
- 默认地址：在父节点添加点击事件 派他思想 传入点击 和所有 将所有的 改为 0 自己改为1

- ```
  @click='changeDefault(item,遍历的数组)'
  changeDefault(item,遍历的数组){
      //遍历的数组全部的isdefault为0
      遍历的数组.forEach(i=>i.isDefault=0)
      item.isDefault=1
  }
  ```

- 根据默认地址动态显示 收货地址 使用find方法
  - 使用find方法 查找数组的符合条件的元素
  - ```
    计算属性(){
        return this.xxx.finde(item=>item.选中状态==1) ||{}
    }
    使用    计算属性.需要的属性
    ```

## 将API接口配置在原型上

- 建议：
  - 当数据存在组件页面自己身上时 使用此方法
  - 无需vuex 保存

```
//引入API
import * as API from '地址'
//同全局事件总线一样
 beforeCreate() {
   Vue.prototype.$API=API
 }
//使用    当数据存在组件页面自己身上时  使用此方法
this.API.请求方法()
```

## 提交订单

- 逻辑：点击结算按钮是 携带信息 提交到服务器  服务器返回订单号
- 成功：保存订单号，路由携带订单号跳转路由

## 支付页面

- 金额： 通过订单号请求服务器 返回数据渲染 （别在生命周期函数 使用await async，生命周期中调用方法）
  - 成功：组件中保存返回数据
- 支付按钮 弹框 使用饿了么ui
- 前端生成二维码：根据后端返回数据
  - NPM i qrcode
  - 使用：

```
import QRCode from 'qrcode'
QRCode.toDataURL('I am a pony!')
  .then(url => {
    console.log(url)
  })
  .catch(err => {
    console.error(err)
  })
```

- 支付按钮事件
  - 弹窗逻辑
    - 生成二维码地址
    - 弹窗ul
      - 需要使用定时器一直询问服务器是否支付完成 setinterval
      - 成功：清楚定时器 保存支付成功返回的code 关闭弹窗 跳转之个人中心路由 支付成功
      -

```
data() {
  return {
    //支付相关信息:支付钱数、订单号、二维码地址
    payInfo: {},
    code: "",
    timer:null
  };
},
   methods: {
  //立即支付按钮
  async open() {
    //生成一个二维码URL
    let url = await QRCode.toDataURL(this.payInfo.codeUrl);
    //第一个参数:即为内容区域
    //第二个参数:标题
    //第三个参数:组件的配置项
    this.$alert(`<img src=${url}>`, "请你微信扫码支付", {
        //弹窗配置
      dangerouslyUseHTMLString: true, //将字符串转换为标签
      center: true, //居中
      showClose: false, //右上角的关闭按钮不显示
      confirmButtonText: "支付成功", //确定按钮的文本
      showCancelButton: true, //显示取消按钮
      cancelButtonText: "支付遇见问题", //取消按钮的文本
      closeOnClickModal: true, //点击遮罩层关闭messagebox

      beforeClose:(action, instance, done)=>{ //在消息盒子关闭之前会触发
         //action参数:可以区分用户点击的是取消【cancel】、确定【confirm】
         //instance参数:当前消息框组件VC
         //done参数: 是一个函数,函数可以关闭消息盒子
         if(action=='confirm' && this.code==200){ //confirm我已支付按钮
            //清除定时器

            clearInterval(this.timer);
```

```
            //关闭盒子
            done();
            //路由跳转
            this.$router.push('/paysuccess');
          }else if(action=='cancel' && this.code!=200){ //cancel第一个按钮
              //清除定时器
            clearInterval(this.timer);
            //关闭盒子
            done();
            this.$message.error('支付遇见问题请联系超管豪哥');
          }
        }
      });
      //查询支付结果,开启定时器每隔一段时间询问支付结果
     this.timer  = setInterval(async () => {
        //发请求获取支付结果
        let result = await this.$http.reqPayResult(this.payInfo.orderId);
        //返回数据当中：code=200代表支付成功  code=205未支付
        if (result.code == 200) {
          //支付成功了
          //存储一下支付成功的code数值，通过他判断支付是否成功
          this.code = result.code;
          //清除定时器
          clearInterval(this.timer);
          //关闭messagebox
          this.$msgbox.close();
          //在路由跳转
          this.$router.push('/paySuccess');
        } else {
          //未支付
          this.code = result.code;
        }
      }, 1000);
    },
    //获取支付信息
    async getPayInfo() {
      let result = await this.$http.reqPayInfo(this.$route.query.orderId);
      if (result.code == 200) {
        this.payInfo = result.data;
      }
    },
  },
```

## 个人中心

- 设置二级路由出口和配置 二级路由 重定向解决 进去没有 子组件显示
- 自己封装的组件：分页器、日历

# 项目最后

## 图片懒加载

- npm i vue-lazyload
- 入口文件 引入 improt VueLazyload from 'vue-lazyload'
- 配置对象：Vue.use(VueLazyload,{loading:atm}) 数据未返回的替换图
- 使用：v-lazy
- 自定义插件

- vue插件一定暴露了一个对象 且 有一个install的方法
- vue.use()插件一使用 就自动调用 install方法

## 表单验证

- vee-validata@2 插件
- 新建一个validate.js 1.引入ivue 2.improt Vuevalidate from 'vee-validate' 3.vue.use
- 入口文件 引入 路径即可 如需暴露 import '路径'
- 提示信息：validate.js
- 使用：input 添加name 和 v-validate
  - 是否一样？ 将正则 改为 is：和谁一样的名字
  -

```
import Vue from  'vue'
import VeeValidate from 'vee-validate'
import zh_CN from 'vee-validate/dist/locale/zh_CN'   // 引入中文 message
Vue.use(VeeValidate)
VeeValidate.Validator.localize('zh_CN', {
messages: {
...zh_CN.messages, //中文信息
is: (field) => ${field}必须与密码相同 // 修改内置规则的 message，让确认密码和密码相同
},attributes: { // 给校验的 field 属性名映射中文名称
phone: '手机号',
code: '验证码',
password:'密码',
password1:'确认密码',
isCheck:'协议'
}})
//基本使用
<input
          placeholder="请输入你的手机号"
          v-model="phone"
          name="phone"
          v-validate="{ required: true, regex: /^1\d{10}$/ }"
          :class="{ invalid: errors.has('phone') }"
        />
<span class="error-msg">{{ errors.first("phone") }}</span>

//自定义规则

//自定义校验规则

//定义协议必须打勾同意
VeeValidate.Validator.extend('名字', {

validate: value => {
```

```
return value
},getMessage: field => field + '必须同意'
})
//我同意使用
<input type='cheackbox'
          placeholder="请输入你的手机号"
          v-model="agree"
          name="agree"
          v-validate="{ required: true,‘名字 ’: true}"
          :class="{ invalid: errors.has('agree') }"
        />
 <span class="error-msg">{{ errors.first("agree") }}</span>

const success = await this.$validator.validateAll(); //当全部验证成功后 才能执行注册
```

## 路由懒加载

- 当打包上线时 js影响页面加载
-
```
//原:
{
    path: '/foo',
    component: Foo
}
//现1:
{
    path: '/foo',
    component: ()=>import('路径')
}
//原型为: 只有一条语句时 可以省略 花括号 和retrun
const 组件名 =()=>{ return import('路径') }
```

## 处理map文件

- npm run  build
- map文件 就是未加密的代码 准确输出哪一行报错　　项目不需要可以去除掉
- vue.config.js 配置：productionSourceMap：false

## 够买服务器

- 腾讯云等
- 安全组：打开端口号
- 利用 Xshell 6 登录服务器 Linux　　xftp 7　上传文件到服务器（linux 可视化）
- Linux指令：/ 根目录
- cd 跳转目录　ls查看目录　root家目录
- mkdir 创建目录　　pwd 查看绝对路径

## nginx反向代理

- 反向代理服务器

- xshell进入根目录/etc  目录下有nginx则进入  无则安装 yum install nginx
- vim  nginx.conf文件
- 主要添加  1、解决服务器如何找到项目

```
location /{
    root   dist 的文件目录;
    index  index.html;
    try_files $uri $uri/ /index.html;
}
```

- 2.反向代理

```
location /api{
    proxy_pass  数据服务器地址
}
```

- service nginx  start 运行nginx服务器

# 复习

## 组件通讯方式

### props

- 父子通信
- 函数形式：本质是子组件给父组件传递数据  updateChecked="hander"  相当于自定义事件
- 非函数：父组件给子组件传递数据  name='myname'
- 书写方式 ['todolist'] {todolist:类型} {todolist:类型,default:[]}

### 自定义事件

- 子给父
- 父使用自定义事件(事件名需要全部小写)  子使用this.$emit  子向父传递数据

### 全局事件总线

- 万能
- beforCreate(){Vue.prototype.$bus=this}
- 通知用 this.$bus.$emit('事件名')
- 接受  this.$bus.$on('事件名',()=>{})

### vuex和pubsub-js: vue不用

### 插槽

- 父子通信 一般结构 通信的是结构
- 默认插槽

- 具名插槽
- 作用域插槽：子组件的数据来源于父亲 子组件决定不了自身结构与外观
    - ```
      //父组件中:
      <list :todos='todos'>
          //子组件结构
          <template slot-scope='变量'>//接受子回传的东西    多个数据时 可以进行解构
              {{变量}}
          </templat>
      数组在父组件 todos
      //子组件中:
       props:{ todos:Array}  //接受父 传递的数据 数组
      <li v-for='(item,index) in todos' :key='index'>
          <slot :todo='item'>  //回传给父组件 对象形式
          </li>
      ```

## event深入

- 原生dom -- button 等可以绑定系统事件 click 等
- 当组件绑定系统事件时 默认为自定义事件 使用.natve 就可以出发系统事件 利用事件委派
- 原生dom 不要书写自定义事件 无意义

## v-model深入

- 数据双向绑定 收集表单数据
- 实现原理： value 和input事件获取当前元素内容 来实现数据双向绑定
- ```
  <input  :value='msg'  @input='msg=$event.target.value' >
  ```
- 可以通过v-model 实现父子组件数据同步
    - v-model ='msg' 相当于 :value='msg' @input='msg=$event.target.value'

## sync

- 可以实现父子组件数据同步
- $event 文档里有写当在父级组件监听这个事件的时候，我们可以通过 $event 访问到被抛出的这个值：
- :money.sync 含义1：父给子传递了一个money数据 给当前子组件绑定了一个自定义事件

```
//无sync修饰  子组件使用props接受 money   点击事件回调使用 $emit('事件名，回调)
<Child :money='money' @changemoney='$event'/>
//子组件使用props接受 money   点击事件回调使用 $emit('事件名，回调)
<Child :money.sync='money'/>

```

## $attrs与$listeners

- 饿了么ul 使用
- $attrs： 属于组件自身的一个属性 可以获取父组件传过来的props数据
  - 如果子组件使用props接受了 则atrrs 就无法获取
  - 使用 v-bind='$attrs'  不同缩写：  用于封装组件  传递的props过度时使用
- $listeners 获取父组件传递的自定义事件
  - 使用v-on='$listeners'

## $children和$parent

- ref可以获取dom节点  也可以获取子组件标签 (操作子组件的数据和方法)
- $children: 获取全部子组件  返回数组  使用forEach遍历
- $parent：获取父组件

## 混入mixin

- 如果项目中很多结构类似功能 逻辑也相似  的js 代码 可以写成一个默认暴露的js文件
- 引入js文件  mixins:[引入的名字] 与data同级