day08

课程复习

- vue的基本指令
- 动态的样式
- 修饰符
- watch监听（侦听器）
- computed （计算属性）
- 生命周期
- 过滤器（局部，全局）
- 过渡动画（内置的状态6）
- 组件的创建（局部，全局）
    - 组件通信
    - 父子组件通信
    - 子父组件通信
    - 非父子组件通信（单一事件管理，vuex状态管理以及离线存储）
- ref属性
- is 动态组件切换
- slot槽口
- 在vue脚手架中如何引入插件，以jq为案例调取接口
- 路由
    - 基本路由的定义

    - 路由的嵌套
    - 路由传参之query
    - 动态路由（一定要修改路由文件）
    - 编程式导航

作业：

- 分析项目
    - 5个二级页面
        - 首页
        - 购物车
        - 购物车结算
        - 个人中心
        - 分类产品列表
    - 4个一级页面
        - 分类
        - 登录
        - 注册
        - 商品详情
- 创建项目并搭建路由

路由高阶

一、路由的模式

- 哈希模式（hash）默认的
- history模式

在路由index.js中去修改mode（模式）这个属性
比如： mode:'history' 或者 mode:'hash'

- hash模式和history 有什么区别

一个带#号，一个不带。这种回答不行！！！过于low

hash和history都可以模拟一段完整的url路由。部署到环境的时候。hash可以前进后退刷新都没有问题，采用window.onhashchange 实现的，那么这种地址方式有点丑，如果你不愿意，你可以采用history模式，采用的是HTML5提供的Interface 中pushState() replaceState()，但是这个模式要想玩要后端（Java或者php）支持，因为这个模式是完整的地址，它可以前进后退，但是不能刷新，因为一刷新就去匹配后端路径，所以需要后端去修改一下服务端配置

二、项目的打包压缩

打包项目 npm run build
会生成一个dist文件

三、模拟后端服务器

- 下载一个express框架

```
npm init -y  创建一个pageage.json

npm install express  安装express创建
```

- 创建一个app.js文件

利用node.js搭建了一个后端服务器

当你要修改app.js的时候，要记得重启（Node服务器）！！！

```
//一、引入express插件
const express = require('express')
//二、调用express方法
const app = express()
//五、引入静态资源
//把前端打包之后的资源部署到搭建好的node服务器上
app.use(express.static('./static'))
//四、创建一个get方式
app.get('/home',(req,res)=>{
    res.send('这么神奇么。。。。')
})
//三、创建一个监听
app.listen(3000,()=>{
    console.log('服务器已经启动，端口号3000~')
})
```

路由导航守卫

比如：进入五方桥基地。
进入到南大门，被保安大爷拦截=>全局拦截
进入6号楼，被6号楼大门的门禁拦截 =>路由独享
进入到104教室，被104的门禁所拦截=>组件拦截，进入之前，组件更新，离开

一、全局导航守卫

- 前置钩子函数 (全局,用的比较多)

beforeEach((to,from,next)=>{

})

全局导航守卫案例之登录拦截

```
router.beforeEach((to,from,next)=>{
    console.log(to,'去哪')
   // console.log(from,'从哪里来')
   // next()
   //一、如果去的是登录页，就放行（next）
  if(to.path=='/login'){
    next()
    return
  }
   // 二、如果它登录的了，就放行（next）
   if(localStorage.getItem('isLogin')){
     next()
     return
   }
   //三、以上逻辑都不是就强制跳转到登录页
   next('/login')
})
```

- 后置钩子函数(几乎不用)

  afterEach((to,from)=>{

  })

二、路由独享守卫

beforeEnter(to,from,next){

}

路由独享守卫案例

```
    {
      path:'/foodList',
      component:foodList,
      beforeEnter(to,from,next){
        console.log(to,'路由独享tootototot')
        console.log(from,'路由独享from')
        //如果来源是home我就让进入美食列表，否则，就全部跳转到首页，即使你知道我的地址也没用
        if(from.path=='/home' || from.path=='/foodDetail'){
          next()
        }else{
          next('/home')
        }
      }
    },
```

三、组件守卫

beforeRouteEnter(to,from,next){

//进入组件之前

}

beforeRouteUpdate(to,from,next){

//进入组件之前

}

beforeRouteLeave(to,from,next){

//进入组件之前

}

组件导航守卫案例

```
 beforeRouteEnter(to, from, next) {
    //当前没有this,因为组件实例未创建
    console.log(this, "当前组件实例！！！");
    console.log(to, "组件进入之前tttotototo");
    console.log(from, "组件进入之前
fromfromfrom");
    if (from.path == "/movieList") {
      next();
    } else {
      next("/home");
    }
  },
  beforeRouteUpdate(to, from, next) {
    //动态路由情况下，去切换你的动态路由值，会触发组件更
新导航守卫
    console.log(this, "更新组件实例");
    console.log(to, "组件更新tttotototo");
    console.log(from, "组件更新fromfromfrom");
    next();
  },
  beforeRouteLeave(to, from, next) {
    //this指向组件实例
    console.log(this, "组件实例啊啊啊啊啊");
    console.log(to, "组件离开tttotototo");
    console.log(from, "组件离开fromfromfrom");
    next();
```

```
}
```

完整的导航解析流程

导航被触发。
在失活的组件里调用 beforeRouteLeave 守卫。
调用全局的 beforeEach 守卫。
在重用的组件里调用 beforeRouteUpdate 守卫 (2.2+)。
在路由配置里调用 beforeEnter。
解析异步路由组件。
在被激活的组件里调用 beforeRouteEnter。
调用全局的 beforeResolve 守卫 (2.5+)。
导航被确认。
调用全局的 afterEach 钩子。
触发 DOM 更新。
调用 beforeRouteEnter 守卫中传给 next 的回调函数，创建好的组件实例会作为回调函数的参数传入。

路由的零碎知识点

当我们做完项目，对于前端人员来说，面临最大的困难就是，优化
图片的加载
数据的加载
打包压缩
。。。。
目的是为了部署服务器的时候，提高渲染速率

一、路由懒加载

```
//路由懒加载，把路由文件切割成模块，当调用函数的时候，才去执行，首次加载未执行

①、 const 变量名= ()=>import('组件地址')
②、 const 变量名 = ()=>Promise.resolve(import('组件地址'))

{
    path:'/地址',
    //component:变量名,
    //或者下列写法
    component: ()=>import('组件地址')
}
```

二、name属性

```
赋值
        {
          path:'/sort',
          component:Sort,
          name:'商品分类'
        },

取值:
this.$route.name

{{$route.name}}
```

三、alias（别名）

```
        {
          path:'/cart',
          component:Cart,
          name:'购物车',
          alias:'/bugbug' //别名
        },
```
用户通过访问path以及alias 都可以渲染Cart这个组件

数据交互（axios）

读音： 阿克西奥斯河

前面阶段学过的数据交互

- 原生ajax
- jq.ajax({})

axios它并不是vue的属性和方法，它是基于promise开发的独立的http状态库

官网：

```
中文文档：http://www.axios-js.com/
英文文档：https://www.npmjs.com/package/axios
```

下载命令:

```
npm install(i) axios
版本号： + axios@0.21.0
```

概念：

一个基于promise的客户端用于浏览器和原生node.js

特点:

从浏览器中创建 XMLHttpRequests
从 node.js 创建 http 请求
支持 Promise API
拦截请求和响应
转换请求数据和响应数据
取消请求
自动转换 JSON 数据
客户端支持防御 XSRF(跨站伪造请求)

使用方法:

引入你下载好的axios库

import axios from axios

get请求

```
一、直接调用axios()
axios({
    url:'你要请求的地址',
    method:'get',//get方式可以省略
    params:{
        name:'',
        data:'',
        age:''
    }
})
.then((res)=>{
    //res 响应结果
```

```
})
.catch((err)=>{
    //err 错误捕获
})
二、直接执行.get()
//没有传参时
axios.get('url路径')
.then((res)=>{
    //res 响应结果
})
.catch((err)=>{
    //err 错误捕获
})
//传入参数
axios.get('url路径',params:{
    //参数对
    age:''
})
.then((res)=>{
    //res 响应结果
})
.catch((err)=>{
    //err 错误捕获
})

//三种方式
async function getUser() {
  try {
  //成功的时候执行
    const response = await axios.get('/user?
ID=12345');
    console.log(response);
  } catch (error) {
```

```
  //捕获错误
    console.error(error);
  }
}
```

post请求

```
//一、直接调用axios()
axios({
    url:'调取的地址',
    method:'post',
    data:{}
})
.then((res)=>{
    //res 响应结果
})
.catch((err)=>{
    //err 错误捕获
})

//调取post方法
//无参数
axios.post('url地址')
.then((res)=>{
    //res 响应结果
})
.catch((err)=>{
    //err 错误捕获
})

//传参
axios.post('url地址',{
    age:'444'
```

```
})
.then((res)=>{
    //res 响应结果
})
.catch((err)=>{
    //err 错误捕获
})
```

脚手架中解决跨域问题

config =>index.js

```
    proxyTable: {//代理服务 设置跨域解决代码
      //  /api只是自定义的，因为我们大部分官方称接口为api
      '/api':{
        target:'http://www.ujiuye.tech:5000',//你要解决的跨域目标
        changeOrigin:true,//是否跨域
        pathRewrite:{
          '^/api':'http://www.ujiuye.tech:5000'
        }
      }
    },
```

接口封装

封装一个接口的模块，用来集中管理项目接口，当项目接口有修改的时候，我们可以快速定位，更重要的是利于我们代码的优化和维护

- 创建一个文件夹，创建一个文件

当前文件用于管理axios核心库的属性的方法

axios.js

```
//引入axios核心库
import axios from 'axios'

//重新创建一个新的axiosAPI
const http = axios.create()

//导出
export default http
```

- 创建一个index.js

  用于管理接口

index.js

```
//引入你封装好的新的axios
import http from './axios'
//封装一个音乐搜索接口
export function getSearch(params){
    return http.get('/api/search',{
        params
    })
}
```

- 应用

```
//引入封装好的接口
import {getSearch} from '../../util/axios'

     //调取音乐搜索接口
     getSearch({
```

```
            keywords:this.val
        })
        .then(res=>{
            console.log(res,'响应')
            if(res.data.code===200){
                this.resultList =
res.data.result.songs
            }
        })
        .catch(err=>{

        })
```

如何运行后端服务器（node搭建）

一、把空表导入数据库

首先链接3306，如果有就双击运行
然后，右键新建数据，自定义名字，选择utf-8
然后双击打开，在表上右键选择运行sql,找到sql文件点开始，成功

二、运行后端服务器

一、解压缩
如果没有Node_modules npm isntall 安装一下
二、修改config =>global.js

```
exports.dbConfig = {
    host: 'localhost',
    user: 'root', //你的用户名
    password: '123',//你的密码
    port: 3306, //你的数据库端口号
```

```
    database: 'ushop' // 你的数据库名字
}
```

三、注释掉app.js中的token拦截
大概 37-46

```
/* app.use(async (req, res, next) => {
    if (!req.headers.authorization) {
        res.send(MError("请设置请求头,并携带验证字
符串"));
    } else {
        if (!await checkToken(req)) { // 过期
            res.send(Guest([],"登录已过期或访问权
限受限"));
        } else {
            next();
        }
    }
}); */
```

四、如果你想修改端口
bin文件=>www文件中，大概15行的位置去修改

```
var port = normalizePort(process.env.PORT ||
'3000');
```

修改之后一定要记得重启！！！

作业

一、整理今天的笔记

二、今天的案例练习3遍

三、填写每日监测项目

四、继续优化小U商城

面试题

hash模式和history模式

> 区别：
> hash模式：
> 1.采用的是window.onhashchange事件实现。
> 2.可以实现前进 后退 刷新。
> 3.比如这个URL：http://www.zhenbang.com/#/index，hash 为#/index。
> 它的特点在于：hash 虽然出现URL中，但不会被包含在HTTP请求中，
> 对后端完全没有影响，因此改变hash不会重新加载页面
> history模式：
> 1.采用的是利用了HTML5 History Interface 中新pushState() 和replaceState() 方法。
> 2.可以前进、后退，但是刷新有可能会出现404的报错
> 3.前端的url必须和实际向后端发起请求的url 一致http://www.haha.com/niu/id 。
> 如果后端缺少对/niu/id 的路由处理，将返回404错误。
> 不过这种模式要玩好，还需要后台配置支持。因为我 用是个单页客户端应用，
> 如果后台没有正确的配置，当用户在浏览器直接访问 http://oursite.com/user/id，这就不好看
> 所以呢，你要在服务端增加一个覆盖所有情况的候选 如

> 果 URL 匹配不到任何静态资源，
> 则应该返回同一个 index.html 页面，这个页面就 app 依赖的页面。