面试题记录

1.盒子居中的方法：（8种）

①弹性盒子居中display: flex；justify-content: center;align-items: center;

②position: absolute;top: 50%;left: 50%;transform: translate(-50%,-50%);

margin-left 和margin-top负自身一半

③postion: absolute;left,right,top,bottom都为0

④行内块display: inline-block;text-align:center;vertical-align: middle

⑤表格display: table-cell;text-align: center;vertical-align: middle;

2.es6 的特有的类型

① let const 两者都有块级作用域② 箭头函数③ 模板字符串④ 解构赋值⑤ for of 循环⑥ import 、export 导入导出⑦ set 数据结构⑧ ...展开运算符⑨ 修饰器 @⑩ class 类继承⑪ async、await⑫ promise⑬ Symbol ⑭ Proxy 代理，classList，Object.assign

es5新增 document.queryselectorAll 。Object.defineProperty,数组map,filter,reduce,reduceRight,some,every

forEach，

watch：

① 可接受两个参数；② 监听时可触发一个回调，并做一些事情；③ 监听的属性必须是存在的；④ 允许异步

watch 配置：handler、deep（是否深度）、immeditate （是否立即执行）

computed：

① 有缓存机制；② 不能接受参数；③ 可以依赖其他 computed，甚至是其他组件的 data；④ 不能与 data 中的属性重复

方案 2（推荐）：该方法是通过$once 这个事件侦听器在定义完定时器之后的位置来清除定时器

```
mounted(){
 const timer = setInterval(()=>{
    console.log(1)
 },1000)
 this.$once('hook:beforeDestroy',()=>{
  clearInterval(timer)
 })
}
```

2.vuex的使用理解

3.排序，算法

冒泡排序

- 遍历元素，跟其下一个元素对比
- 把最大的逐个往后排列

  var arr = [12,3,44,343,55,1,23]; for(var i=0;i<arr.length-1;i++){ for(var j=0;j<arr.length-i-1;j++){ if(arr[j]>arr[j+1]){ var current = arr[j]; arr[j] = arr[j+1]; arr[j+1] = current; } } } console.log(arr);

选择排序法

- 把当前元素分别跟后面所有的元素对比
- 把最小的逐个往前排列

  var arr = [12,3,44,343,55,1,23]; for(var i=0;i<arr.length;i++){ for(var j=i+1;j<arr.length;j++){ if(arr[i]>arr[j]){ var current = arr[i]; arr[i] = arr[j]; arr[j] = current; } console.log("666"); } } console.log(arr);

快速排序法

```
/*
* 利用递归函数实现排序
* 每次获取数组中间元素cItem
* 把大于和小于cItem的元素分别放置在arrGt和arrLt两个数组中，
* 利用concat组合递归调用函数返回的值
* 直到数组的长度小于1时，直接返回原数组调出递归
*/
var arr = [10, 8, 20, 5, 6, 30, 11, 9];
function fastSort(arr){
    //6. 递归退出条件
    if(arr.length<=1){
        return arr;
    }
    //1. 找出数组中间位置元素
    var cIdx = parseInt(arr.length/2);

    //2.删除中间元素，避免与自己本身进行对比而造成死循环
    var cItem = arr.splice(cIdx,1);//[6],arr=[10, 8, 20, 5, 30, 11, 9]

    //3. 创建两个空数组用于保存大于或小于cItem的数字
    var arrLt = [];//[5]
    var arrGt = [];//[10,8,20,30,11,9]

    // 4.遍历数组，分别与cItem进行对比
    // 把小于cItem的数写入arrLt
    // 把大于cItem的数写入arrGt
    for(var i=0;i<arr.length;i++){
        if(arr[i]<cItem[0]){
            arrLt.push(arr[i])
        }else{
            arrGt.push(arr[i]);
```

```
        }
    }
    // 5.组合排序后的数组
    return fastSort(arrLt).concat(cItem,fastSort(arrGt));
}
console.log(fastSort(arr));
```

sort排序（字符串编码默认）

4.跨越的解决方法

nginx jsonp（get,script标签，全局函数) 服务器代理 orgin * ，，跨站脚本攻击, postMessage)，web sockets来跨域

iframe(容易引起xxs【Cross Site Scripting 】

①使用window.name+iframe来进行跨域②通过document.domain+iframe来跨子域(只有在主域相同的时候才能使用该方法)

https://www.cnblogs.com/lcspring/p/11079754.html

5.单点登录或者cookie在主系统登录，不同域名的子系统，如何自动登录。

前端ajax请求设置设置crossDomain（true)与withCredentials(true)的作用 : //允许携带cookie，能带cookie到后台的， //同时后台也能往前台写cookie

如果要把Cookie发到服务器，一方面要服务器同意，指定`Access-Control-Allow-Credentials`字段 （true)后台

后端**Access-Control-Allow-Origin(http://localhost:8080)** : 控制允许携带Cookie访问的域 （指定具体的域，父域)

Access-Control-Allow-Credentials（true)： 允许客户端携带证书式访问

例子

e.g前端

$.ajax({ type: 'GET',url: "http://www.cookie.test.com:8080/web2/ajaxServlet",data: {"username":"haha"},dataType:"json", crossDomain: true, xhrFields: { withCredentials: true }, success: function (data){console.info(data); } });

e.g后端 const cors = require('cors');

var corsOptions = { origin: 'http://localhost:8080',

credentials: true, maxAge: '1728000' //这一项是为了跨域专门设置的}app.use(cors(corsOptions))

Access-Control-Allow-Credentials: true

6.vue-cli的跨越解决，（配置方法）

- http-proxy-middle-ware 里面配置
- 例如：我们当前主机为http://localhost:3000/，现在我们有一个需求，如果我们请求/api，

- 我们不希望由3000来处理这个请求，而希望由另一台服务器来处理这个请求怎么办?
- 现在，我们利用express在3000端口启动了一个小型的服务器，
- 利用了app.use('/api', proxy({target: 'http://localhost:3001/', changeOrigin: true}));
- 这句话，使发到3000端口的/api请求转发到了3001端口。即请求http://localhost:3000/api相当于请求http://localhost:3001/api。

7.浏览器的渲染（dom的构建）

- https://www.cnblogs.com/peteremperor/p/6285449.html

8.vue的全家桶Vue +vue-router+vueX+elementUi

9.react 的全家桶，React+React-router+React-redux+Axios+ant-design.

10.小程序的生命周期

```
 onLaunch() {
  console.log('onLaunch监听小程序初始化');
}

onShow() {
  console.log('onShow监听小程序显示');
}

onHide() {
  console.log('onLaunch监听小程序隐藏');
}
 onLoad(options) {
    console.log('onLoad监听页面加载');
  }

  onReady() {
    console.log('onReady监听页面初次渲染完成');
  }

  onShow() {
    console.log('onShow监听页面显示');
  }

  onHide() {
    console.log('onHide监听页面隐藏');
  }

  onUnload() {
    console.log('onUnload监听页面卸载');
  }
```

11.bookstrap

12.网站如何优化

①代码优化（Eslint） ②按需加载，（如路由懒加载，页面触底懒加载，ui框架按需引入） ③合并压缩（共用一张精灵图，css，js合并） ④图片优化（大小）

⑤合并多次请求

svg,base64的图片。

缓存：针对静态资源（图片，css,js） 更新文件通过添加时间戳（其实就是hash值）； 通过配置文件，webspack.config.js.引入 cleanWebpackPlugin= require('clean-wepack-plugin') output: 文件名加 -[hash]- 在 Plugins:[//加入 new cleanWebpackPlugin(); new wepackplugin{ hash: }]

⑤ueo 界面友好性，操作便捷性

⑥seo 语义化标签。ssr

⑦cdn引入

13.组件更新了，视图不更新，是因为缓存问题。

添加时间搓避免缓存。

针对静态资源（图片，css,js） 更新文件通过添加时间戳（其实就是hash值）；

14.vue双向绑定原理：

通过 Object.defineProperty()来劫持各个属性的 setter,getter，在数据变动时发布消息给订阅者，触发相应的监听回调

15.钩子函数。

16.vue如何优化首页的加载速度?

①第三方js库CDN引入

②vue-router路由懒加载

③压缩图片资源

④静态文件本地缓存。

⑤服务器端渲染。 koa框架（引入ejs模板引擎）。对应模板template.js用<%=title %> (用%号包裹，可以遍历)

16.vue和react的区别是什么?

① React严格上只针对MVC的view层,Vue则是MVVM模式

② virtual DOM不一样,vue会跟踪每一个组件的依赖关系,不需要重新渲染整个组件树.而对于React而言,每当应用的状态被改变时,全部组件都会重新渲染,所以react中会需要shouldComponentUpdate这个生命周期函数方法来进行控制

③ 组件写法不一样, React推荐的做法是 JSX + inline style, 也就是把HTML和CSS全都写进JavaScript了,即'all in js'; Vue推荐的做法是webpack+vue-loader的单文件组件格式,即html,css,jd写在同一个文件;

④ 数据绑定: vue实现了数据的双向绑定,react数据流动是单向的

⑤ state对象在react应用中不可变的,需要使用setState方法更新状态;在vue中,state对象不是必须的,数据由data属性在vue对象中管理

17.$nextTick用过吗，有什么作用?

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。

解决的问题：有些时候在改变数据后立即要对dom进行操作，此时获取到的dom仍是获取到的是数据刷新前的dom，无法满足需要，这个时候就用到了$nextTick。

18 vue模板中为什么以_、$开始的变量无法渲染?

名字以 _ 或 $开始的属性不会被 Vue 实例代理，因为它们可能与 Vue 的内置属性与 API 方法冲突。用 vm.$data._property 访问它们。

18.怎么实现路由懒加载呢?

第一种（最常用）：

```
const Foo = () => import('./Foo.vue')
const router = new VueRouter({
  routes: [
    { path: '/foo', component: Foo }
  ]
})
```

第二种:

```
const router = new Router({
  routes: [
   {
     path: '/index',
     component: (resolve) => {
        require(['../components/index'], resolve) // 这里是你的模块 不用import去引入了
     }
    }
  ]
})
```

axios.封装的ajax.

19.axios怎么解决跨域的问题?

使用axios直接进行跨域访问不可行，我们需要配置代理 （服务器请求服务器的方法）

1.配置BaseUrl

```
import axios from 'axios'
Vue.prototype.$axios = axios
axios.defaults.baseURL = '/api'  //关键代码
```

2.配置代理

在config文件夹下的index.js文件中的proxyTable字段中，作如下处理：

```
proxyTable: {
 '/api': {
   target:'http://api.douban.com/v2', // 你请求的第三方接口
   changeOrigin:true,
// 在本地会创建一个虚拟服务端，然后发送请求的数据，并同时接收请求的数据，
//这样服务端和服务端进行数据的交互就不会有跨域问题
   pathRewrite:{  // 路径重写，
    '^/api': ''
// 替换target中的请求地址，也就是说以后你在请求http://api.douban.com/v2/XXXXX
//这个地址的时候直接写成/api即可。
   }
  }
}
```

\1. 在具体使用axios的地方，修改url如下即可

```
axios.get("/movie/top250").then((res) => {
  res = res.data
  if (res.error === ERR_OK) {
    this.themeList=res.data;
  }
 }).catch((error) => {
  console.warn(error)
})
```

因为我们给url加上了前缀/api，我们访问/movie/top250就当于访问了：localhost:8080/api/movie/top250（其中localhost:8080是默认的IP和端口）。

在index.js中的proxyTable中拦截了/api,并把/api及其前面的所有替换成了target中的内容，因此实际访问Url是api.douban.com/v2/movie。

20 说说你对proxy的理解?

Proxy用于修改某些操作的默认行为，也可以理解为在目标对象之前架设一层拦截，外部所有的访问都必须先通过这层拦截，因此提供了一种机制，可以对外部的访问进行过滤和修改。

```
var demo = new Proxy(target, logHandler)
demo.name  //name被读取
var proxy = new Proxy(target, handler);
```

21.vue怎么实现强制刷新组件?

答：① v-if：methods中reload函数，通过this$nextTick。改变this.update=true来显示刷新 ② this.$forceUpdate。直接在reload函数使用

22 vue中怎么重置data?

this.$data获取当前状态下的data，this.$options.data()获取该组件初始状态下的data。

然后只要使用Object.assign(this.$data, this.$options.data())就可以将当前状态的data重置为初始状态。

23.跟keep-alive有关的生命周期是哪些？描述下这些生命周期

activated和deactivated两个生命周期函数

①.activated：当组件激活时，钩子触发的顺序是created->mounted->activated

②.deactivated: 组件停用时会触发deactivated，当再次前进或者后退的时候只触发activated

24.怎么解决vue动态设置img的src不生效的问题?

主要是用require('./....png');因为动态添加src被当做静态资源处理了，没有进行编译，所以要加上require

```
<img class="logo" :src="logo" alt="公司logo">
data() {
  return {
    logo:require("./../assets/images/logo.png"),
  };
}
```

25 vue 生命周期，各个阶段简单讲一下?

在new Vue实例化后会自动执行初始化函数，会初始化事件，生成vue实例的整个生命周期，这个时候就有整个生命周期

breforeCreate（）：实例创建前，这个阶段实例的 data 和 methods 是读不到的。

created（）：实例创建后，这个阶段已经完成数据观测，属性和方法的运算，watch/event 事件回调，mount 挂载阶段还没有开始。$el 属性目前不可见，数据并没有在 DOM 元素上进行渲染。

created 完成之后，进行 template 编译等操作，将 template 编译为 render 函数，有了 render 函数后才会执行

beforeMount（）：在挂载开始之前被调用：相关的 render 函数首次被调用

mounted（）：挂载之后调用，el 选项的 DOM 节点被新创建的 vm.$el 替换，并挂载到实例上去之后调用此生命周期函数，此时实例的数据在 DOM 节点上进行渲染

后续的钩子函数执行的过程都是需要外部的触发才会执行

有数据的变化，会调用 beforeUpdate，然后经过 Virtual Dom，最后 updated 更新完毕，当组件被销毁的时候，会调用 beforeDestory，以及 destoryed。

26 部分面试题

①虚拟dom。 diff算法 （带有dom节点的js对象）

②http的进程和线程

③elementui 和iview的区别

④对象里面加 静态属性 static 有什么作用的。

⑤\ jwt 分别代表什么》？ 后端返回一些敏感信息给你，你如何解决。反序列化问题。

解答：JWT由三段构成，中间用小数点隔开 header.payload.signature

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ`

JWT的构成和流程

```
一。构成
头部header中明文存放类型和加密算法，并通过json格式base64编码后保存
     *{"typ":"JWT", "alg":"HS256"} //示范的jwt header头
载荷payload中可自定义存放用户信息，一样用json格式base64编码保存。
    *//一个正儿八经的payload示范
{
    "iss": "appid_xxxxxx"
    "sub": "012345122",
    "exp": "1572246721840",
    "iat": "1592246721840"
}
签名signature是签名信息，这个部分需要base64加密后的header和base64加密后的payload使用.连接组成的字符串，然后通过header中声明的加密方式进行加盐secret组合加密，这就构成了jwt的第三部分。
于签名部分采用不可逆算法，且secret仅服务端可知，这就保证了签名部分攻击者无法伪造，即便修改了payload内容，也无法通过signature签名校验。
 *signature签名流程:将header和payload进行base64编码，然后使用"."将header和payload拼接起来
    *String base64data = Base64.encode(header)+"."+Base64.encode(payload)
A.JWT可以防XSS吗?
答：不可以。JS可以通过JSON.stringify(localStorage)获取到存储在浏览器中的JWT。攻击者可以利用XSS漏洞页面获得受害者的JWT令牌。

BJWT可以防CSRF吗?
答：这个要分情况。CSRF利用的原理是浏览器针对跨站请求会携带目标站的Cookie。所以如果JWT放到了Cookie里面，那是无法御防CSRF攻击的。如果JWT单独放到HTTP的Header头里面，由于跨站请求无法携带这些附加的Header，就可以起到防御CSRF攻击的目的。

注意点：
1．信息泄露
不应该在JWT的payload部分存放敏感信息，比如用户的密码、隐私信息等，因为该部分是客户端可解密的部分。一旦JWT被攻击者拿到，这些信息也将随之泄露
2．密钥保护
```

保护好secret私钥，该私钥非常重要。因为如果攻击者知道了secret，就可以随意伪造payload并自行生成合法的signature签名。JWT就无法防篡改防伪造，整个验证机制也就土崩瓦解

https://mp.weixin.qq.com/s?src=11&timestamp=1607480537&ver=2755&signature=sB2fz4LxPnfVw92jS2d20tFOvvW2Up2f8P3qcvozhiixxdWW6bSc47veBPhRZbjevPeds6Ltz750suG7amRqJs9mHLo62SBWqo1g6Ys4Nkr1Lu9r-Q3vR18E*u8*KS*w&new=1
二。流程：
客户端将用户名/密码通过某种加密的方式发送给服务器。
服务器接收到客户端请求后进行验证，验证通过服务器生成token返回给客户端。客户端将token存储在本地。
客户端每次请求将token携带在http header头中， 服务器端将token取出进行解密。
三。优点：
服务器端无需保存token，以加解密的方式代替存储，节省了内存空间。（时间换空间）
无状态的token不依赖于服务器保存会话信息，更利于水平扩展。
相比于传统的session认证方式，jwt对移动端的支持更友好。

⑥spa 首屏优化， 这部分首屏的css 会阻塞js的样式 ， 如何解决 css 的异步加载方法

首先，这个问题就涉及浏览器的渲染过程：

1. 加载 `HTML` 资源
2. 解析 `HTML`
3. 加载 `CSS` 资源，同时构建 `DOM树`
4. 解析 `CSS` ，同时渲染 `DOM树`
    - 方法一：利用媒体查询--设置一个当前浏览器不支持的值
        - `<link rel="stylesheet" href="./index.css" media="none" onload="this.media='all'">`
        - 浏览器将会异步加载这个 `CSS` 文件（优先度比较低），在加载完毕之后，使用 `onload` 属性将 `link` 的媒体类型设置为 `all` ，然后便开始渲染。最后渲染成 `<link rel="stylesheet" href="./index2.css">`
    - 提前加载资源

      `<link rel="preload" href="./index.css" as="style">` 用到的时候，浏览器便从缓存中拿取

27.讲一下虚拟dom的更新过程

28 面试部分问题集合

1. 深浅拷贝有几种方式?

   深 JSON.parse(JSON.stringfy(obj)); 浅（Object.assign{{},obj}

   {...obj}

   for(let key in obj){

   newObj[key]=obj[key] 1

   }
2. Js数据类型有哪些?
3. 数据去重有哪些方法?

4. v-if和v-for优先级?

   v-for比较高

5. 反向代理和正向代理的区别?

   正向代理:

   正向代理通过上面的图理解其实就是用户想从服务器拿资源数据，但是只能通过proxy服务器才能拿到，所以用户A只能去访问proxy服务器然后通过proxy服务器去服务器B拿数据，这种情况用户是明确知道你要访问的是谁，在我们生活中最典型的案例就是"翻墙"了，也是通过访问代理服务器最后访问外网的。

   反向代理:

   反向代理其实就是客户端去访问服务器时，他并不知道会访问哪一台，感觉就是客户端访问了Proxy一样，而实则就是当proxy关口拿到用户请求的时候会转发到代理服务器中的随机（算法）某一台。而在用户看来，他只是访问了Proxy服务器而已，典型的例子就是负载均衡了。

6. 页面渲染?
7. 如何用webpack优化项目?
8. 如何理解继承?
9. 如何理解作用域?
10. 闭包的优点是什么
11. bug:① 添加了hash之后，会导致改变文件内容后重新打包时，文件名不同而内容越来越多，因此这里介绍另外一个很好用的插件clean-webpack-plugin。②引用动态设置图片的src不生效，轮播图用elementUi有bug,用require('./....png');

12.什么是自动化部署?

如果用gitLab做git的服务器的话，可以配置commit的钩子函数，实现自动部署和线上发布，就相当于服务器监听你的提交，在你commit之后，服务器自动执行以下npm run build ,放到对应的测试服务器目录配置文件在根目录下有.gitlab-ci.yml文件，起作用的是下边的一段代码，script是linux脚本，拷贝文件到指定的静态资源CDN目录和web服务器目录，这块知识点是gitlab-ci持续集成，可以关注一下，svn应该也有类似的配置，让运维帮忙配一下。

1. `npm-build-test:`
2. `image: cdn路径`
3. `stage: build`
4. `cache:`
5. `untracked: true`
6. `paths:`
7. `- node_modules/`
8. `before_script:`
9. `- export BI_ENV="test"`
10. `script:`
11. `- "npm install --registry=http://代理地址 --sass_binary_site=https://npm.taobao.org/mirrors/node-sass/"`
12. `- "npm run build"`
13. `- "rsync -auvz dist/index.html  ip::服务器开发分支目录/trunk/resources/views/index/"`
14. `- "rsync -auvz dist/* 静态资源cdn目录/trunk/bi/"`
15. `only:`
16. `- master  分支名称`

29.组件之间的传值?

1.父组件与子组件传值

父组件传给子组件：子组件通过props方法接受数据; 子组件传给父组件：$emit方法传递参数

2.非父子组件间的数据传递，兄弟组件传值

Bus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。项目比较小时，用这个比较合适。

子组件向父组件传递数据

//子组件通过$emit方法传递参数

uni.$on uni.$off uni.$once uni.$emit （用uni.app尽量用他的接口提供的能更好兼容多端)

在onLoad 监听， 在onUnload 卸载。

如果返回跳转可以取到数据（因为A页面已经创建，返回了原先的页面，如果A,跳转到B，数据比较多，用uni.$emit跳转 ，打开B页面，

这时B页面还没有打开，不能注册到监听事件uni.$on或者uni.$once会报错。

如果页面没有打开，将不能注册监听事件 uni.$on 和 uni.$once 。

如果需要给未打开的页面传参，使用url传参，或者使用vuex

3.provide和inject

祖元素provide同data写法： provide（）{return {form:this},

孙元素inject 同props写法： inject：['from']

4.$listeners和$atrrs

```
vue.2.4，祖元素template
...fu.vue
<child :foo="foo"  :coo='coo"' @getChildItem="getData"></child>
...  js {
    methods:{    getData(res){console.log('res',res)}}
}
...child.vue
template
    <div>
        {{foo}}
         {{$attrs}}
        <childItem v-bind="$attrs" v-on="$listeners ></childItem>
    </div>
```js
  {
      props:['foo'],
      inheriAttrs: false,
  }
````childItem
```

```
    template
         <button @click="handleClick></button>
```js
   {
       methods:{
           handleClick(){
               this.$emit('getChildItem','childmsg')
           }
       }
   }
```

5.this.dispatch和this.broadcast

elementui的广播模式

引入minxin

import emitter from 'element-ui/src/mixins/emitter';

componentName: 'ElFormItem',

mixins: [emitter],

mounted(){ this.dispatch('ElForm', 'el.form.addField', [this]);} //formItem

this.broadcast('ElTimeSelect', 'fieldReset', this.initialValue);

this.$on('el.form.blur', this.onFieldBlur);

this.$on('el.form.change', this.onFieldChange);

```
./src/mixin.js
function broadcast(componentName, eventName, params) {
  this.$children.forEach(child => {
    var name = child.$options.componentName;
    if (name === componentName) {
      child.$emit.apply(child, [eventName].concat(params));
    } else {
      broadcast.apply(child, [componentName, eventName].concat([params]));
    }
  });
}
export default {
  methods: {
    dispatch(componentName, eventName, params) {
      var parent = this.$parent || this.$root;
      var name = parent.$options.componentName;

      while (parent && (!name || name !== componentName)) {
        parent = parent.$parent;

        if (parent) {
          name = parent.$options.componentName;
        }
      }
```
```

```
      if (parent) {
        parent.$emit.apply(parent, [eventName].concat(params));
      }
    },
    broadcast(componentName, eventName, params) {
      broadcast.call(this, componentName, eventName, params);
    }
  }
};
```

30 $route和$router的区别

$route是路由信息对象，包括，，，，，，等路由信息参数。而route是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。而$router是“路由实例”对象包括了路由的跳转方法，钩子函数等.

31.vue.js的两个核心是什么?

数据驱动 组件系统

32.数据劫持:

采用数据劫持结合发布者-订阅者模式 的方式，通过 Object.defineProperty（） 来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调。

```
①直接调用$router.push 实现携带参数的跳转
②父组件中：通过路由属性中的name来确定匹配的路由，通过params来传递参数。
③父组件：使用path来匹配路由，然后通过query来传递参数
```

33.react 的虚拟DOM，diff算法，为什么这么火?

渲染遍历时要加入key,用以区分，比如两个人同名字，也有身份证我们就能很快的区分出来，如果没有key我们要费更多的时间区分。

34.vue和react 的区别。

35.封装过vue-cli吗? sass封装原理

36.vue3.0最新的东西，proxy,Ts，react最新的东西

37 .vue插件的封装吗?

1.封装过vue-router（如封装一个购物车. 定义一个函数 let Luyou = (Vue,option)=>{

Vue.component('router-tiao',{

template:""

})

},

2.使用插件 Vue.use(router-tiao) ElementUI 也是Vue.use(ElementUI)

3.监听hash window.onhashchange 函数， 获取hash= window.location.hash.slice(1) // \

原生插件需要 用函数里面 install属性

install

38.sass与less的区别：

sass和less都是css的预编译处理语言 ,mixins，参数，嵌套规则，运算，颜色，名字空间，作用域，JavaScript赋值等 加快了css开发效率 ,都可以配合gulp和grunt等前端构建工具使用

sass和less主要区别:在于实现方式 less是基于JavaScript的在客户端处理 所以安装的时候用npm，sass是基于ruby所以在服务器处理。

39.值类型，引入ts的void.

①void 空值 某种程度上来说，void类型像是与any类型相反，它表示没有任何类型。 当一个函数没有返回值时，你通常会见到其返回值类型是void

```
function warnUser(): void {
    alert("This is my warning message");
}
```

声明一个`void`类型的变量没有什么大用，因为你只能为它赋予`undefined`和`null`：

```
let unusable: void = undefined;
```

②null 和undefined:

TypeScript里，`undefined`和`null`两者各自有自己的类型分别叫做`undefined`和`null`。 和`void`相似，它们的本身的类型用处不是很大：

```
// Not much else we can assign to these variables!
let u: undefined = undefined;
let n: null = null;
```

默认情况下`null`和`undefined`是所有类型的子类型。 就是说你可以把`null`和`undefined`赋值给`number`类型的变量。

然而，当你指定了`--strictNullChecks`标记，`null`和`undefined`只能赋值给`void`和它们各自。 这能避免*很多*常见的问题。 也许在某处你想传入一个`string`或`null`或`undefined`，你可以使用联合类型`string | null | undefined`。 再次说明，稍后我们会介绍联合类型。

> 注意：我们鼓励尽可能地使用`--strictNullChecks`，但在本手册里我们假设这个标记是关闭的。

40.导航钩子有哪几种，分别如何用，如何将数据传入下一个点击的路由页面?

① 全局导航守卫

前置守卫

```
router.beforeEach((to, from, next) => {
  // do someting
});
```

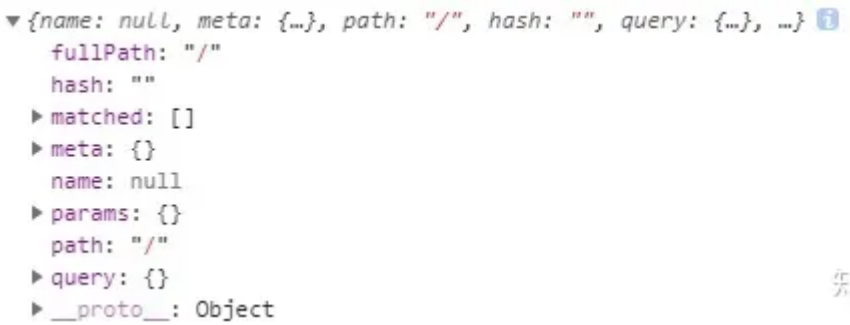
后置钩子（没有 next 参数）

```
router.afterEach((to, from) => {
  // do someting
});
```

② 路由独享守卫

```
cont router = new  VueRouter({
 routes: [
  {
    path: '/file',
    component: File,
    beforeEnter: (to, from ,next) => {
       // do someting
    }
   }
 ]
});
```

顺便看一下路由里面的参数配置：



③ 组件内的导航钩子

组件内的导航钩子主要有这三种：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave。他们是直接在路由组件内部直接进行定义的

beforeRouteEnter

```
data(){
 return{
   pro:'产品'
 }
},
beforeRouteEnter:(to,from,next)=>{
  console.log(to)
  next(vm => {
   console.log(vm.pro)
  })
}
```

注：beforeRouteEnter 不能获取组件实例 this，因为当守卫执行前，组件实例被没有被创建出来，我们可以通过给 next 传入一个回调来访问组件实例。在导航被确认时，会执行这个回调，这时就可以访问组件实例了

仅仅是 beforRouteEnter 支持给 next 传递回调，其他两个并不支持，因为剩下两个钩子可以正常获取组件实例 this

40.如何通过路由将数据传入下一个跳转的页面呢?

答：params 和 query

①params

```
传参
this.$router.push({
 name:"detail",
 params:{
   name:'xiaoming',
 }
});
接受
this.$route.params.name
```

query

```
传参
this.$router.push({
  path:'/detail',
  query:{
    name:"xiaoming"
  }
 })
接受 //接收参数是this.$route
this.$route.query.id
```

那 query 和 params 什么区别呢?

① params 只能用 name 来引入路由，query 既可以用 name 又可以用 path（通常用 path）

41.axios的封装

其实也可以其实也可以全局设置拦截，我们用的是 `axios`，在 `main.js` 中引用，设置**根路径、状态码、token、超时时间**等全局设置，代码如下:

```
import axios from'axios'
// axios配置
Vue.prototype.$http =axios
// 配置默认axios参数
axios.defaults.baseURL ='/'
axios.defaults.timeout =1000000
axios.defaults.headers.post['Content-Type']='application/x-www-form-urlencoded;charset=UTF-8'
// 添加请求拦截器

axios.interceptors.request.use(function (config) {
   let token =localStorage.getItem('token')
   if(token== null && router.currentRoute.path =='/login'){
// 本地无token,不为登录 跳转至登录页面
   router.push('/login')
   }else{
   if(config.data==undefined){
     config.data ={"token":token}
     }
else{
   Object.assign(config.data,{"token": token})

    }
}
  return config
},function(error){
    iView.Message.error('请求失败')
    returnPromise.reject(error)})

// 返回结果拦截
axios.interceptors.response.use(function (response){
  if(response.hasOwnProperty("data")&&typeof response.data =="object"){
      if(response.data.code ===998){
// 登录超时 跳转至登录页面
      iView.Message.error(response.data.msg)
          router.push('/login')
          return Promise.reject(response)
    }else if(response.data.code === 1000){
// 成功
        return Promise.resolve(response)
    }
}else{
    return Promise.resolve(response)
}
function(error){
// 请求取消 不弹出
  if(error.message !='0000'){
    iView.Message.error('请求失败')
```

```
}
// 请求错误时做些事
return Promise.reject(error)
})
```

42.监听的方法

当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的

1.外部监听生命周期函数

定时器的清除 this.$once('hook:beforeDestory',=>{balabala})

①优点：//随时监听随时取消 编辑表单，保存按钮禁用，结束编辑时取消监听

```
<template>
  <custom-select />
</template>
<script>
export default {
    components:  {
        CustomSelect
    },
    watch: {
         //①formData对象写法
        formData : {
         handle(newVal,oldVal){
        // 不能用 if  (newVal !==oldVal) 判断 //执行要做的事情
        },
        immediate:  true //加这个可以初始化页可以拿到，理解执行属性
        deep: true //深度监听，表单的任意一项修改，变更为被修改状态，（每一项就需要深度监听）
        }
    },
    //② 普通写法
    searchVal(newVal,oldVal){
       //balabla
    }

   },
    beforeCreate(){
       const unWatch=this.$watch('formData', ()=>{
      //做你要做的事情 console.log(''ddd');
        }, {
       deep: true
       })
       unWatch()//执行取消监听
    }
}
</script>
```

2.外部监听生命周期函数

```
<template>
  <custom-select @hook:updated="$_handleSelectUpdated" />
</template>
<script>
export default {
    components:  {
        CustomSelect
    },
    methods: {
        $_handleSelectUpdated(){
            console.log('custom-select组件的updated钩子函数被触发)
        }
    }
}
</script>
```

43.项目较小可利用vue.2.6 vue.observable手动打造一个Vuex

```
file: store.js
<script>
export const store = Vue.observable({
    userInfo: {}
    roleIds: []
})
export const mutations = {
    setUserInfo(userInfo){
        store.userInfo = userInfo
    },
    setRoleIds(ro)
}
</script>

/.../
<template>
<div>{{userInfo.name}}</div>
</template>
<script>
    import {store,mutations} from '../store
    export default {
        computed: {
            userInfo() {
                return store.userInfo
            }
        },
        created () {
            mutations.setUserInfo( {
                name: '子君'
```

```
            })
        }
    }
</script>
```

44.vue的函数组件写法

函数组件,不能通过$emit对外暴露事件，调用事件只能通过context.listeners.click的方式调用外部传入事件

```
<template>
<div>{{userInfo.name}}</div>
</template>
<script>

    export default {
        functional: true,
        props: {
            avatar: {
                type:String
            }
        },
       render(h,context) {
           const {props} = context;
           if(props.avatar) {
               return <img src={props.avatar}></img>
           }else {
               return <img src="default-avatar.png"></img>
           }
       }
    }
</script>
```

45.本地服务mock搭建

https://blog.csdn.net/pma934/article/details/90748226

1安装 koa-generator脚手架

npm install koa-generator -g

2一键生成koa2项目

koa2 myProject

3安装mockjs

`npm install mockjs --save`

①在routes文件中引入增加 const Mock = require('mockjs')

router.prefix('/clients') 【请求字段换名字)

ctx.body = Mock.mock(generateData({ }) 【generateData,格式化响应数据】;

②app.js中引入 const clients = require('./routes/clients') ，调用

app.use(clients.routes(), clients.allowedMethods())

4.h5项目中使用

manifest.json 中配置代理就可以使用了

```
"h5": {
        "devServer": {
            "port": 8088,
            "disableHostCheck": true,
            "proxy": {
                "/api": {
                    "target": "http://127.0.0.1:3000/",
                    "changeOrigin": true,
                    "secure": false,
                    "pathRewrite": {
                        "^/api": ""
                    }
                }
            }
        },
        "router": {
            "mode": "history"
        },
        "optimization": {
            "treeShaking": {
                "enable": true
            }
        }
    }
```

46.对于nginx的理解和使用

①nginx --本身就是代理，不需要配置CORS.conf文件。

node 设置允许访问的域名是 http://127.0.0.4:1902/

前端vue启动地址是 http://127.0.0.4:1902/ （登录密码访问，对应上成，改成http://127.0.0.3:1902/，请求错误

但是如果开启了nginx，前端改成http://127.0.0.3:1902/，本地host 127.0.0.3 my.wangzhan.com

就可以成功。（本身就是服务代理没有跨域之说) [E:\personnal\nginx-1.16.0]--详情查看本地文件

```
简化 sendfile        on;   #<--开启cdn获取分布式网络静态资源->
upstream zp_server1  { #<--zp_server1 自定义服务器变量名->
     server localhost:19011;
}

server {
    location / [
```

```
        proxy-pass http://main #跳转upstream的main ;
    ]
    location /router {
        proxy-pass http://router 跳转 upstream的router ;
    }
   # location /api {
      # proxy_pass http://127.0.0.3:1901/api;  #接口转发请求
   #}
}
```

②实例

```
修改本地host(src)
C:\Windows\System32\drivers\etc\host
#10.108.81.42     www.test.csair.com
127.0.0.3           my.wangzhan.com
<--nginx 配置修改  文件 nginx.conf --->

#user  nobody;
worker_processes  1;

# error_log  logs/error.log;
error_log  logs/error.log  notice;
error_log  logs/error.log  info;

pid        logs/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  logs/access.log  main;
    rewrite_log on;

    sendfile        on;

    #keepalive_timeout  0;
    keepalive_timeout  65;
    tcp_nodelay on;

    #gzip  on;

    #设定实际的服务器列表
```

```
 upstream zp_server1{
    # server 127.0.0.3:1901;
    server localhost:19011;
}

server {
    listen       80;
    # server_name web.example.com;
    server_name  127.0.0.3 my.wangzhan.com;

    #index index.html
    root E:/study-nan/jiuxian-peizhi/jiuxianwang;

    # charset koi8-r;
    charset utf-8;

    access_log  logs/host.access.log  main;

    #代理配置参数
    proxy_connect_timeout 180;
    proxy_send_timeout 180;
    proxy_read_timeout 180;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarder-For $remote_addr;


    #反向代理的路径（和upstream绑定），location 后面设置映射的路径
    location / {
        # proxy_pass http://127.0.0.1:1901/$1?$args;
        proxy_pass http://127.0.0.3:1901;
        include       CORS.conf;
    }

    location /api {
        proxy_pass http://127.0.0.3:1901/api;
    }
    location /jiuxian {
        proxy_pass http://127.0.0.3:1901/jiuxian;
    }


    #静态文件，nginx自己处理
    location ~ ^/(images|javascript|js|css|flash|media|static)/ {
        root E:/study-nan/jiuxian-peizhi/jiuxianwang/dist;
        #过期30天，静态文件不怎么更新，过期可以设大一点，如果频繁更新，则可以设置得小一点。
        expires 30d;
    }

    #设定查看Nginx状态的地址
    location /NginxStatus {
        stub_status           on;
        access_log            on;
        auth_basic            "NginxStatus";
```

```
            auth_basic_user_file  conf/htpasswd;
        }

        #禁止访问 .htxxx 文件
        location ~ /.ht {
            deny all;
        }
    }
}
```

注意点：vue 编译后的dist本地借助自身项目启服务。需要运行本地项目，上线项目则不需要。proxy也会失效，所有

dist和后端代码会共同放在一个根目录下，都是静态文件。

```
listen       80;
        server_name  127.0.0.3 my.wangzhan.com;
location / {

            # proxy_pass http://127.0.0.1:1901/$1?$args;

            # proxy_pass http://127.0.0.3:1901;

            root E:/study-nan/jiuxian-peizhi/jiuxianwang/dist;

            index index.html index.htm;

            try_files $uri $uri/ @router;
  }

  <---->
真实环境网上例子
location / {
            root   D:\code\base-platform-vue\dist;#项目打包后的目录地址
            index  index.html index.htm;
            try_files $uri $uri/ @router;#需要指向下面的@router否则会出现vue的路由在nginx中刷新出现
404
        }
        #对应上面的@router，主要原因是路由的路径资源并不是一个真实的路径，所以无法找到具体的文件
        #因此需要rewrite到index.html中，然后交给路由在处理请求资源
        location @router {
            rewrite ^.*$ /index.html last;
        }


   最近总洁，（前端编译好的可以直接dist拷贝里面的到nginx的html 注释#html，重新建html进去，外网就可以访问了)(node拷贝到c盘桌面，用supervisor开启服务）（server 可以写多个没有影响）
   server {
        listen       80;
        server_name  localhost:8888; /或者（120.17.14.220：40）没有购买域名，外网的前端地址

        location /dev-api/ {
```

```
            proxy_pass http://127.0.0.1:8090/; (服务器上，后端地址)
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
        location /weibo/ {
            proxy_pass http://www.sina.com/data.json/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    }

    <------->
    另外 /ajax/
     upstream test{
        server 10.3.133.168:1000 weight=1;
        server 10.3.133.168:2000 down; //不做负载
        server 10.3.133.123:2000 backup; //备份（其他挂了用他）
        server 10.3.133.168:3000 weight=2; 优先
    }

    location /frontend{
        alias kerwinhtml/frontend
    }

    location /api/ { //双斜杆是请求路由 但斜杆是浏览器上的路径路径名
            proxy_pass http://127.0.0.3:1901/api;
            # proxy_pass http://127.0.0.4:1902/api;
        }
    node 压缩编译
    添加这个
}

    // 静态资源服务器（
app.use(express.static('./'))

// 启动gzip压缩
const compression = require('compression');
app.use(compression());

app.use((req,res)=>{
    // 任何的请求返回index.html的内容
    fs.readFile(path.resolve(__dirname,'index.html'),(err,data)=>{
        res.set('Content-Type','text/html; charset=utf-8')
        res.send(data);
    })
})

app.listen(1911,()=>{
    console.log(`Server is runing on port 1911`)
})
```

③安装mysql, 安装注册机navicat ,然后安装wavm,里面要勾选mysql,然后配置环境,

C:\wamp64\bin\mysql\mysql8.0.18\bin

npm包安装 npm i mysql

```
<template>
  <custom-select @hook:updated="$_handleSelectUpdated" />
</template>
<script>
export default {
    components:  {
        CustomSelect
    },
    methods: {
        $_handleSelectUpdated(){
            console.log('custom-select组件的updated钩子函数被触发)
        }
    }
}
</script>
```

47.react 相关

A. react+dva

```
dva中的每个model，实际上都是普通的JavaScript对象，包含(有点类似 store)
- namespace
- state     -- data
- reducers  -- emit state (纯函数)
- effects   --  *  (call,put)  call --请求  put --emit reducer来更新状态。(可以多个put)
- subscriptions
- 项目按照 npm i mysql
********
数据流向
数据的改变发生通常是通过用户交互行为或者浏览器行为（如路由跳转等）触发的，当此类行为会改变数据的时候可以通过 dispatch 发起一个 action，如果是同步行为会直接通过 Reducers 改变 State ，如果是异步行为（副作用）会先触发 Effects 然后流向 Reducers 最终改变 State，所以在 dva 中，数据流向非常清晰简明，并且思路基本跟开源社区保持一致（也是来自于开源社区）。
********

app.model({
  namespace: 'count',
  state: {record: 0,current: 0,},
  reducers: {
    add(state) {
      const newCurrent = state.current + 1;
      return { ...state,
        record: newCurrent > state.record ? newCurrent : state.record,
        current: newCurrent,
      };
    },
```

```
    minus(state) {
      return { ...state, current: state.current - 1};
    },
  },
  effects: {
    *add(action, { call, put }) {
      yield call(delay, 1000); ------------delay 是 封装的请求函数，1000为其参数
      yield put({ type: 'minus' });
    },
  },
  subscriptions: {
    keyboardWatcher({ dispatch }) {
      key('⌘+up, ctrl+up', () => { dispatch({type:'add'}) }); -- Subscription 语义是订阅，用于订阅
一个数据源，然后根据条件 dispatch 需要的 action。
    },
  },
});
```

①.创建新应用

```
一.创建新应用
$ npm install dva-cli -g
创建新应用
$ dva new dva-quickstart
$ cd dva-quickstart
$ npm start

使用 antd
$ npm install antd babel-plugin-import --save

编辑 .webpackrc，使 babel-plugin-import 插件生效。

{
+  "extraBabelPlugins": [
+    ["import", { "libraryName": "antd", "libraryDirectory": "es", "style": "css" }]
+  ]
}
```

②定义路由 （路由组件）

```
定义路由 （路由组件
①新建 (route component) routes/Products.js
import React from 'react';

const Products = (props) => (
  <h2>List of Products</h2>
);
export default Products;

添加路由信息到路由表，编辑 router.js :
        + import Products from './routes/Products';
        ...
```

```
        + <Route path="/products" exact component={Products} />
② 组件同样写componet
```

③ dva 通过 model 的概念把一个领域的模型管理起来

```
三 dva 通过 model 的概念把一个领域的模型管理起来，包含同步更新 state 的 reducers，处理异步逻辑的
effects，订阅数据源的 subscriptions 。
export default {
  namespace: 'products',
  state: [],
  reducers: {
    'delete'(state, { payload: id }) {
      return state.filter(item => item.id !== id);
    },
  },
};
```

④定义model

```
四 定义model   model/product.js
export default {
  namespace: 'products',
  state: [],
  reducers: {
    'delete'(state, { payload: id }) {
      return state.filter(item => item.id !== id);
    },
  },
};
 --- namespace 表示在全局 state 上的 key
 --- state 是初始值，在这里是空数组
 --- reducers 等同于 redux 里的 reducer，接收 action，同步更新 state

 dva 通过 model 的概念把一个领域的模型管理起来，包含同步更新 state 的 reducers，处理异步逻辑的
effects，订阅数据源的 subscriptions 。
export default {
  namespace: 'products',
  state: [],
  reducers: {
    'delete'(state, { payload: id }) {
      return state.filter(item => item.id !== id);
    },
  },
};
```

⑤⑥，connect连接起来 具体调用

```
五 app.model(require('./models/products').default);
```

六　具体里面调用，平常的react 和redux 连接是用 react-reducer
①这里具体页面调用也是需要连接，dva 提供了connect 方法
② 触发用dispatch({type:'product/delect,payload:id})　-- product[[namespace,同key]] --delect [[reducers 里的 delect 方法]]--- 参数 state,修改state

```
import { connect } from 'dva';
import ProductList from '../components/ProductList';
export default connect(({ products }) => ({
            products,
}))(Products);
```

印象中，页面的数据和state的连接起来

附件 https://dvajs.com/guide/develop-complex-spa.html#%E5%A4%9A%E4%BB%BB%E5%8A%A1%E8%B0%83%E5%BA%A6
dva是redux-saga的封装，也是可以使用这种操作的。使用take操作进行事件监听

```
A  const [result1, result2]  = yield all([
  call(service1, param1),
  call(service2, param2)
])
```

B 把多个要并行执行的东西放在一个数组里，就可以并行执行，等所有的都结束之后，进入下个环节，类似promise.all的操作。一般有一些集成界面，比如dashboard，其中各组件之间业务关联较小，就可以用这种方式去分别加载数据，此时，整体加载时间只取决于时间最长的那个。

react, 数据设置 state本身未定义，可以 直接 this.setState({ isDidMount: false,});,

在render 函数能打印到 state,数据， 在 周期和 constructor里面打印不到更新，以为他用了nextTick，异步更新

请求最好在componentDidMount （官方推荐，也可以 willMount,

> 在react中，通过富文本编辑器进行操作后的内容，会保留原有的标签样式，并不能正确展示。

> 在显示时，将内容写入__html对象中即可。具体如下：

```
<div dangerouslySetInnerHTML = {{ __html: checkMessages.details }} />
```

> react原理剖析

```
react 做了哪些事情
借助babel   tranform-babel
转成成 jsx
jsx---转成 React.createElement, ---转成 虚拟dom
一个js对象
dom  --- 通过 ReactDom.render,,--直接操作dom,不好，缓存些东西
<template>
</template>



ReactDom.render (
    //上面,
```

```
    //let template = React.createElement('div',null,
    //   React.createElement('p',{name:'guozhou',age:18}))
    template
     (document.querySelector('#app'), 就是取dom节点啊
)



proxy 放在的一个返回对象，调用熟悉和传值。
React.createElement('div',{name:'guozhou',size:'122'},hellword)

实际用了 new Proxy() 实际返回了一个对象，
我们执行用这个对象的属性 如 proxyData.div
执行例子 (proxyData.div) ({name; 'guozhou'},'hellwold');
function createElement (target,prop,...childrens) {
        return proxyData = new Proxy({},{
        get (target,tagName) {
         return (attrs={},...childrens)=>{
             //创建节点
             var  self_elem =document.createElement(tagName);
             var self_attrs = attrs;
             var self_childrens = childrens;
             //self_bindAttrs({attrs,elem}) //绑定上去属性节点和值 也是一个函数还有情况为空值情
况等
             //self_addChildrens({childrens,elem})
             //插入和创建文本节点
             return  self_elem
         }
        }
        set () {

        }
        })[prop](props,...childrens)

}


而ReactDom.render 其实也是一个函数

export default {
    render (template,el) {
        el.appendchild(template);
    }
}
```

B .create-react-app my-learn-react

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
```

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

//service worker 是在后台运行的一个线程，可以用来处理离线缓存，消息推送，后端自动更新

// 等任务。serviceWorker.js就是为ract项目注册了一个service worker,用来做资源的缓存，这样

//你下次访问时，就可以更快的获取资源，而且因为资源被注意，serviceWorker只在生产环境

// 中生效(process.env.NODE_ENV === 'production')
```

react项目 行内样式 style={{height:"100px"}}

变量 对象 {对象} className="home" ;

htmlFor = 'labelid'

> //类型检查

```
PropsPage.propTypes ={
  data: PropTypes.string.isRequired, //字符串且必须传
  isHide: PropTypes.bool.isRequired //布尔值且必须传
  age:(props,propName,comName)=>{
      //字定义校验规则，age小于18岁就报错
      if(props[propName]<18){
          return new Error('小于18岁')
      }
  }
}
// PropsTypes.array, PropsTypes.bool
// PropsTypes.func  PropsTypes.number
// PropsTypes.object
// PropsTypes.string  PropsTypes.symbol
//设置默认值
PropsComponent.defaultProps = {
    name: "牛奶糖", //有配置用配置，没有才用默认配置
}

可以异步拿数据
this.state = {
    total: 10
}
this.setState({
    total: 20
},()=>{
console.log('this.state.total',this.state.total)
)
```

```
add = ()=>{
    console.log('这种方式绑定')
}

<input type="text" defaultValue={} onChange={e=>{console.log('e',e.target.value)}}>
<input type="text" value={this.state.count} onChange={e=>{console.log('e',e.target.value)}}>

移动端事件，保存在 event(e) e.touches[0]
onTouchStart onTouchMove onTouchEnd
封装 toast 原理 函数组件， appdend.child
React.Dom(<组件名>,div)
删除  延迟350， 动画效果 0.3s
弹窗结束  done，如果存在，则执行
03尚未
```

umi +dva+react 项目

import router from "umi/router";

router.push router.push(`/information/detail?id=${nearby[0].id}`)

数据层，仓库 model, 服务层，请求services

触发 方法是 put ---触发reducer effect(*) --- to reducers

react yield put({ type: "save", payload: resultData });

vue dispatch('mutations') action---to mutations;

c.react的总洁 ---插槽

```
react的插槽 ，内部传值标签内<div name="chuanzhi"></div>
标签再且套标签就是子组件传的，存在 props.children.
标签内传值 存再 prop.name;
父组件
<zi-button>传值到子组件的内容<button>
子组件 可以用用单括号
<template>

        <div>{this.props.children}</div>
</template>

vue 是用 <slot></slot>
react 3种通信方式 props，自定义事件 插槽
```

D.react的学习总洁 ---hook

```
函数组件无state,无componentDidMount,componentDidUpdate
import React, { useState, useEffect } from 'react';
用 [count, setCount] = useState(0)  ---设置数据state  count,  设置方法 setCount,  初始值 -0;
用 放在同一个执行函数里面
useEffect(() => {})   --- 里面同 componentDidMount,componentDidUpdate

npm install eslint-plugin-react-hooks --save-dev
只在最顶层使用 Hook
只在 React 函数中调用 Hook
--props-changes componentwillReceiveProps
---state-changes shouldComponentUpdate
----生命周期 --componentDidUpate ---vue -watch的效果;
--componentDidMount --发送请求ajax;
```

import {lazy,Suspense} from 'react;

E.react的16版本以上, 懒加载 js chunk切割性能优化 netWork-js查看效果

```
npm i -g create-react-app
npm i antd@4.54 -S
npm i babel-plugin-import -S
npm run eject
//npm i antd-mobile -S
//
["import",{
          "libraryName": "antd",
          "libraryDirectory": "es",
          "style": "css"},'ant'
],
["import",{
          "libraryName": "antd-mobile",
          "libraryDirectory": "es",
          "style": "css"},'ant-mobile'
]

创建项目: create-react-app 项目名称暴露webpack配置信息:npm run eject
安装依赖:

redux router axios antd node-sass
新增目录:  src
    assets 静态资源
    common css - test.module.css

npm i babel-plugin-import
"babel": {
    "plugins": [
      [
        "@babel/plugin-proposal-decorators",
        {
          "legacy": true
        }
      ],
```

```
      [
        "import",
        {
          "libraryName": "antd",
          "libraryDirectory": "es",
          "style": "css"
        }
      ]
    ],
    "presets": [
      "react-app"
    ]
  }
```

①切片使用

```
import {lazy,Supsense} from 'react
let AllCheck= lazy(()=>import './AllCheck'));
// import {add} from 'match';
import('./match).then(match=>match.add());//16版本的新特性;
//fallback 必须传;
render (){
    return (
    <div>
    <button>jiaz</button>
    {this.state.isShow && <Suspense fallback="加载中..." >
     <AllCheck />
    </Suspense>}
    </div>
    )
}
```

②react配置暴露

```
-- react暴露配置 npm run eject  会编译出config文件夹，webpack.config.js --package.json  - eject 会消失， 配置其他的暴露出来
--- proxy 代理配置
fvServer.config.js
proxy:{
      "/dev-api":{
        target: "http://120.76.247.5:2002",
        changeOrigin: true,
        pathRewrite: {
          "^/dev-api":"",
        }
      },
},
--css 模块化-  同vue scoped
asset 文件 创建 -common -public.css
list.module.css
list.js <  import Styles from '../asset/list.module.css;  .reds {background:yellow};
 html     <div className={Styles.reds}>样式文件</div>
```

另一种使用片段的方式是使用 React.Fragment 组件，React.Fragment 组件可以在 React 对象上使用。 这可能是必要的，如果你的工具还不支持 JSX 片段。 注意在 React 中， <></> 是 <React.Fragment/> 的语法糖。

```
<React.Fragment key={bianlian}/>
```

<></> 空标签不可以放key。

<li onClick={goto('/list')>首页

<Switch>
<Route path="/good/:id" component={}>
</Switch>
Route 同vue的 route-view

③各个文件调用情况
main.js
<HashRouter>
<App/>
</HashRouter>,
document.querySelector('#app');
④
app.js
(安装react-router-dom)
(import { Route, Switch,Redirect,withRouter,HashRouter} from 'react-router-dom';)
(@withRouter 或者 export default withRouter(App>)
withrouter作用是用来跳路由的，hasHRouter在withRouter 外面

```
@WithRouter
class App extends React.Component {
    constructor(){super();this.state.pages=[{path:'/home',name: '首页'}]},
    render() {
        return (
        <ul>
        {this.state.pages.map(item=><li key={item} onClick={this.goto.bind(this,item,path)>
{item.name}</li>)}
        </ul>
        <Switch>
                <Route path="home" component={Home}></Route>
                <Redirect from="/" to="/home" ></Redirect>
                { *或者  <Route path="/" exact render={()=>
              <Redirect to="/login" />
            }></Route> *}
        </Switch>
        )
    }
}
export default withRouter(App);
```

@withRouter 装饰器不生效问题
yarn add @babel/plugin-proposal-decorators
将package.json中的babel依赖修改为：

```
"babel": {
    "plugins": [
      [
        "@babel/plugin-proposal-decorators",
        {
          "legacy": true
```

```
        }
      ]
    ],
    "presets": [
      "react-app"
    ]
  },
```

react 16后，只需要 安装react-router-dom
©路由原理
App.js其实待Switch 就相当于vue的router.js (router-view) Route 属性 path={} component={} render={}
可以封装router.js放在注入 app.js。
用切片可以优化性能 代替switch 调用权限路由 <AuthRoute path="/user" component={User}></AuthRoute>

```
let Home = lazy(()=>import('../components/Home'));
let TestHook = lazy(()=>import('../components/TestHook'));
let MovePage = lazy(()=>import('../components/MovePage'));
let Login = lazy(()=>import('../pages/Login'));
let User = lazy(()=>import('../pages/User'));
let IndexPage = lazy(()=>import('../pages/IndexPage'));
 <Suspense fallback={<div></div>}>
            <Route path="/testhook" component={TestHook}></Route>
            <Route path="/home" component={Home}></Route>
            <Route path="/movepage" component={MovePage}></Route>
            <Route path="/login"  component={Login}/>
            {/* <Route path="/user"  component={User}/> */}
            <AuthRoute path="/user" component={User}></AuthRoute>
            <Route path="/indexpage"  component={IndexPage}/>
            <Route path="/" exact render={()=>
              <Redirect to="/login" />
            }></Route>
          </Suspense>
```

路由跳转
push(), replace go goBack goForward
页面刷新问题
用 search：传参； this.props.history.push({

```
    pathname: '/detail',
    search: "?id="+id+'arr='+arr,
})

let str = this.props.location.search
let {itemlist} =localParams(str).search
let arr = JSON.parse(itemlist);
return(
<div>
{arr.map(item)=><p key={item.gid}>商品：{item.gid}-{decodeURIComponent(item.name)}</p>}
</div>
)
```

//易错点 NavLink 用to
Route 用path
重点：监听路由变化

```
function App(props){
    props.history.listen((link)=>{
```

```
        console.log('link',link)
    })
}
hook 是react中16.7新增的一个特性 主要是用来让无状态的组件可以使用状态
```

F.react权限路由控制 permission.js

vue和react的路由history模式下刷新报错的问题 。

```
解决，vue和react模式下刷新，问题（刷新他会去找真实路径的/user,而实际上这个是路由不是真实路径，nginx部署也是根据文件夹跳转的。所有express要配静态，自动跳转到 2002/index.html，刷新就不会出问题.>
const express = require('express');
const fs = require('fs');
const path = require('path');
const app = express();
//静态资源服务器
app.use(express.static('./'))
app.use((req,res)=>{
    //任何的请求返回index.html的内容
    fs.readFile(path.resolve(__dirname,'index.html),(err,data)=>{
        res.set("Content-Type",'text/html; charset=utf-8');
        res.send(data)
    })
})
app.listen(2002,()=>{
    console.log('server is running on 20002);
})
```

G.react的学习总洁 ---react-redux redux 的使用

npm install redux --save //redux插件，

npm install react -redux -- save //包裹组件，注入redux

```
安装
引入
准备state和action
装入仓库
注入到react根组件里面
引入connect
调用store的数据和方法action
````index.js
import {createStore} from 'redux';
import {Provider} from "react-redux";
//准备state 和方法 action
console.log('Provider',Provider)
let defaultState = {
  count: 1
}
function counterReducer(state=defaultState,action){
  return state;
```
```

```
}
//store={store}  必须是store属性否则会报错
        <Provider store={store}>
       <App />
    </Provider>,
let store = createStore(counterReducer);
```app.js
<HashRouter>
        <RouterView/>
</HashRouter>
```IndexPage
import {connect} from "react-redux";

add(){
    this.props.dispatch({type:"",data:{count:++this.state.count}})
}
render(){
    return (
    <div>
    <button onClick={this.add.bind(this)}+1</button>
    </div>
    )
}

export default connect((state)=>{
  return {
    state:state
  }
})(IndexPage);

dispatch 是通过redux的import {connect} from "react-redux";
连接，调用出来的
Reducer
```

react生命周期的坑
1.最新生命周期的变化
https://www.jianshu.com/p/b331d0e4b398
2.生命周期坑点：-7种
①，getDerivedStateFromProps 用于编写反模式代码，受控组件与非受控组件区分模糊，
② componentwillMount --已经被标记弃用，不推荐使用,主要是新的异步渲染架构会导致被多次调用。请求和实际绑定代码应该被移到componentDidmount中
③ componentWillRecieveProps --同样被标记弃用被getDerivedStateFromProps取代性能问题。
④ shouldComponentUpdate 通过 true/false 来确定是否触发重新渲染，主要用于性能优化。
⑤ compnentWillUpDate 因为react新的异步渲染机制，而被弃用，不推荐使用。--- React 开启异步渲染模式后原先的逻辑可以通过 getSnapshotBeforeUpdate与componentDidUpdate改造使用。
⑥ componentWillUnmount 忘记解除事件绑定和去掉定时器清理任务。
⑦ 如果没有添加边界处理。发生异常，用户将会看到一个白屏。
react的请求要放在哪里 --对于异步请求 应该放在componentDidMount中--从时间顺序看，除compenntDidMount可以在a,constructor: 可放，但设计而言不推荐，主要用于初始化state与函数绑定，不承载业务逻辑且随着类属性流行，constructor已很少用。

b componentWillMount：已被标记废弃，在新的异步渲染架构下会触发多次渲染易发生bug,不利于为了react的升级和代码维护。

react性能优化

共同点：
　　场景：当页面数据发生（不管是不是跟之前的值一样，只要是赋值，都是数据发生改变）的时候，一般会导致整个页面（包括组件）进行重新渲染，相当消耗性能
　　作用：可以在相同数据下，或者子组件没有依赖这个修改的数据时避免组件或者页面被重新渲染，都是浅层比较，不适合引用数据改变时的情况
...
区别：
 a,shouldComponentUpdata:
  ①只能在Component类中使用，
  ②可以将上一次的state，props和更新后的state，props进行比较
  ③如果返回true，则重新渲染，否则不渲染

b,React.PureComponent:
  ①使用了pureComponent不可以使用shouldCompoent
  ②自带props，state浅层比较逻辑，没有shouldComponent那么灵活，可以手写逻辑
  ③如果想在引用类型更新的时候重新渲染组件，可以借助forceUpdate

c,React.memo:
 ①只能在函数式中使用
 ②只能对比props
 ③支持第二个参数，可以修改比较过程
 ④如果props相等，‘areEqual'会返回true,如果不相等，则返回false。这与shouldComponentUpdate方法的返回值相反

e.g:
一
```
class Foo extends Component {
  shouldComponentUpdate(nextProps,nextState){
    if(nextProps.count===this.props.count){ //传入的count与组件当前props的count比较,count没改变,return false，不渲染
      return false    //不渲染
    }
    return true;  //渲染
  }
  render() {
    console.log("组件渲染");  //可以看到，当父组件的name改变时，子组件不会打印，只有count改变，才会打印，优化性能
    return null
  }
}
```
二
```
class Foo extends PureComponent {
  render() {
    console.log("组件渲染");
    return null
  }
}
```

```
三
const Foo = memo(function Foo(props) {
  console.log("组件渲染");
  return <div>count:{props.count}</div>
})


import {pureComponent,component} from 'react';
React.PureComponent 通过props和state的浅对比来实现 shouldComponentUpate()。
通过遍历对象上的键执行相等性，并在任何键具有参数之间不严格相等的值时返回false。 当所有键的值严格相等时返回true,
缺点：可能会因深层的数据不一致而产生错误的否定判断，从而shouldComponentUpdate结果返回false，界面得不到更新
```

H.redux-thunk的用法

```
npm i redux-thunk
import {createStore,combineReducers,applyMiddleware} from 'redux'
import counterReducer from "./counter";
import userReducer from "./users";
import thunk from "redux-thunk";
let store = createStore(
   combineReducers({
       counter:counterReducer,
       users:userReducrer
   }),
   applyMiddleware(thunk) // applyMiddleware(thunk,aaa,bbb)
   //express中间件的扩充方法
   //redux-thunk就是redux的一个中间件，就是让redux支持异步请求。类似于(redux-saga)
   https://www.jianshu.com/p/51c8eaa9fa2a
)
```

J.react-ts 用法

```
安装 依赖
npm i -g create-react-app
create-react-app typescript-react-basic --typescript
启动就可以
create-react-app版本问题
create-react-app typescript-react-app --typescript
js- ts
jsx-tsx
.d.ts
src 新建文件 pages/home  Home.tsx; (rcc快捷键 )
interface IProps {

}
interface IState {
    color: string
}
```

```
export default class Home extends Component<IProps,IState> {
    constructor(props:IProps) {
        super(props);
        this.state= {
            color: 'red'
        }
    }
    render(){
        return (
        <div>
                <span style={{color:this.state.color}}>中国加油</span>
                <br>
         </div>
        )
    }
}
```

48.webpack的用法

基础用法

```
作用:
        1.优化:  工程化。vue,react cli
        2.打包:  将多个文件进行压缩，打包成一个文件。带宽。
        3.转换:  es6,ts,jsx,less,sass ;
        webpack 可以import但是解构es6模式不可出现
目标:
        1.可能，通过webpack对公司现有的项目进行从0开始的搭建。
        2.可能对 vue，react脚手架进行添加与修改的操作
webpack当中的一些原理，流程
        先全局安装
        npm install webpack webpack-cli -g
        npm install webpack@4.30.0 -g
        0配置。不需要额外的配置。默认将src下的index.js进行打包，打包到dist ->
一. webpack的结构:
        1.入口:  entry
        2.出口:  output
        3.插件:  plugins
        4.devServer
        5.转换 :  loader
        6.module
        7.mode: 模式 --- =>> 开发模式  生产模式
        配置
        安装:
                webpack: webpack 的核心模块
                webpack-cli: 来执行webpack相关的命令行
                cnpm install webpack webpack-cli -g
                wepack -v 查看当前安装的
                webpack     打包
                webpack --mode development
```

Awebpack：命令
--mode：指定打包模式 development production
B将多个文件打包一个：demo4
webpack src/one.js src/two.js --mode development
C修改输出的文件名字或地址：
--output：指定文件地址
webpack ./src/one.js ./src/two.js --mode development --output haha.js
路径名需要相对位置才可以的 ./src
输出位置指定 ：webpack ./src/one.js ./src/two.js --output haha.js（暂时不生效此命令行）
webpack.config.js
D.自定义配置文件 my.config.js 执行方法：webpack --config my.config.js
package.json
“script": {
"build": "webpack --config my.config.js"
}
配置
package.json
npm init -y
{
"script": {
"build": "webpack src/one.js src/two.js --output my/haha.js --mode development"
}
 }
通过 npm run build 进行打包，相当于执行webpack src/one.js src/two.js --output my/haha.js --mode development
方法二：
//webpack 基于node，首先要有node环境。//webpack --mode development
webpack.config.js:
配置
module.exports = {
mode: "development", //--mode
entry: "./src/mo.js" 指定入口文件
output: {
filename: "haha.js", // 指定输出文件名称
// filename: "my/haha.js" //输出 dist/my/haha.js 在my的文件夹里面
}
}

clean-webpack-plugin html-webpack-plugin
1.下载 npm install clean-webpack-plugin -D
2. 使用 import {CleanWebpackPlugin} from "clean-wepack-plugin"
热更新原理
安装
两步就可以 npm install webpack-dev-server -g 全局安装
执行 webpack-dev-server 命令就可以了

devServer：可以通过属性设置webpack当中的服务。
vue react 开发与生产环境
两个内容：
服务代理。proxy->
地址重定向 。 --->》 单页面应用。
devServer:{
open:true //是否再浏览器当中自动打开。

```
}

等同于  package.json
{
"scripts": {
        "start": "webpack-dev-server"
}
}

module 模块
        module:{
        rules:[
                {
                test:/.\.css$/,
                //npm install style-loader css-loader -D
                loader:["style-loader","css-loader"],
                },
                {
                test:/.\.less$/,
                //npm install less-loader less -D
                loader: ["style-loader","css-loader","less-loader"]
                },
                {
                test:/.\.scss$/,
                loader:["style-loader","css-loader","sass-loader"]
                },
                {
                        // npm install url-loader -D (依赖 file-loader)
                test:/.\.(png|gif|jpg)$/,
                //另外一种写法  loader:["url-loader"]
                use:[
                {
                loader:"url-loader,
                //还可以额外传递一些值，多配置
                query:{
                        limit: 1251118, //limit，图片大小，
                        outputPath:"img"//指定图片放置的文件夹
                }
        }
        ]
        }
]
}
```

自身的优化：生产环境 treeShaking，不用代码不打包 scoped-hosting 作用域提升，作用域更少，代码可读写更高，a+b，只打印3

二，

速度 happypack多子进程打包

体积 ignorePlugin 引入moment，将不需要的语言去掉-避免引入无用模块

ignorePlugin 直接不引入代码中没有

noParse 引入但不打包

```
        避免库的重复打包  module: {
        noParse:[/react\.min\.js$/],
        }
        优雅  scss-loader less-loader
        抽离公共代码块
        webpack的优化:
        用时间搓更新文件命名，导致文件越来越多，用clean-webpack-plugin
        url-loader  query: {limit: 10000,outputPath: '../tupian/lujing'}
webpack.definePlugin 可以注入环境变量
plugins: new webpack.definePlugin({
    ENV: JSON.stringify('production');
})
DllPlugin   动态链接库插件
```

webpack4的特性

```
1.mode属性
开发模式:  a.浏览器调试工具b.注释、开发阶段的详细错误日志和提示
c.快速和优化的增量构建机制
生产模式: a.开启所有的的优化代码，b.更小的bundle大小，
c.去除掉只再开发阶段运行的代码 d.scope hosting 和 Tree-shaking
2. 插件和优化
webpack4删除了CommonsChunkPlugin插件，它使用内置API，optimization.splitChunks和
optimization.runtimeChunk,这意味着webpack会默认为你生产共享的代码块。
其他插件的变化:
   .NoEmitOnErrorsPlugin 废弃 -->使用optimization.noEmitOnErrors 替代
   .ModuleConcatenationPlugin 废弃---》 使用optimization.concatenateModules替代
   .NamedModulesPlugin 废弃 ---》使用optimization.nameModules替代
   . uglifyjs-webpack-plugin 升级到了v1.0版本。

 3. 开箱即用webAssembly
  webAssembly(wasm)会带来运行时性能的大幅度提升，由于在社区的热度，webpack4对他做了开箱即用的支持。你可
以直接对本地的wasm模块进行import或者export操作，也可以通过loaders来
直接import C++、c或者Rust
4.支持多种模块类型:
共5种:
.javascript/auto: 在webpack3里，默认开启对所有模块的系统的支持，包括CommonJs、
AMD、ESM
.javascript/esm:只支持ESM这种静态模块。
.javascript/dynamic:只支持CommonJs和AMD这种动态模块
json:只支持json数据，可以通过require和import来使用。
webassembly/experimental:只支持wasm模块，目前处于试验阶段。
5.oCJs
oCJS的含义是0配置。webpack4受Parcel打包工具启发，尽可能让开发者运行项目的成本变低。为了做到0配置，
webpack4不再强制需要webpack.config.js作为打包的入口配置文件了，他默认的入口为"./src/'和默认出
口'./dist',对小项目而言是福音。
6.新的插件系统
webpack4对插件系统进行了不少修改，提供针对插件和钩子的新API.变化如下:
。所有的hook由hooks对象统一管理，他将所有的hook作为可拓展的类属性。
。当添加插件时，必须提供一个插件名称。
。开发插件时，可以选择sync/callback/promise作为插件类型。
。可以通过this.hooks = {myHook: new SyncHook{...}}来注册hook了。
```

注意点：当使用webpack4时，确保使用Node.js的版本 》= 8.9.4。
因为webpack4使用了很多JS新的语法，他们在新的版本的V8里经过了优化。
webpack index.js -o output_test.js
7.搭建vue webpack
npm install vue-cli -g
vue init webpack hello_vue
npm run start (cnpm install 不行就重新安装依赖)
cnpm install webpack@^4 -D
cnpm install webpack-dev-server@lastest
cnpm install webpack-cli@lastest -D
cnpm install html-webpack-plugin@latest -D
cnpm install eslint-loader@lastest -D
cnpm install vue-loader@latest -D
cnpm installl css-loader@lastest -D
webpack.dev.conf.js (添加mode:"development")
weebpack.prod.conf.js(添加 mode:"production")

```
module.exports = {
  //...
  optimization: {
    splitChunks: {
      chunks: 'async',
      minSize: 20000,
      minRemainingSize: 0,
      maxSize: 0,
      minChunks: 1,
      maxAsyncRequests: 30,
      maxInitialRequests: 30,
      automaticNameDelimiter: '~',
      enforceSizeThreshold: 50000,
      cacheGroups: {
        defaultVendors: {
          test: /[\\/]node_modules[\\/]/,
          priority: -10,
          reuseExistingChunk: true
        },
        default: {
          minChunks: 2,
          priority: -20,
          reuseExistingChunk: true
        }
      }
    }
  }
};
```

npm run build
(cnpm install extract-text-webpack-plugin@lastest -D --不匹配webapck)
npm run build
cnpm install extract-text-webpack-plugin@next -D
很多问题 ，要改。

49.大屏项目搭建react

大屏项目搭建 流程
1npm install -g create-react-app
create-react-app big-screen
npm install @ant-design/charts ant axios redux react-deux redux-thunk react-router-dom
sass-loader style-loader css-loader
http-proxy-middleware
2暴露配置 npm run eject
3建立代理 setupProxy.js
```
const {createProxyMiddleware} = require('http-proxy-middleware');
module.exports = function(app) {
  app.use(
    '/jeecg-boot',
    createProxyMiddleware({
      target: 'http://172.16.24.170:11300',
      changeOrigin: true,
    })
  )
}
```
4.创建store 触发 store --- dispatch (传入 type : payload) -- action(执行返回 {type,payload) -0
从而实现触发store
index.js
```
        import {createStore,combineReducers,applyMiddleware,compose} from  'redux';
        import thunk from 'redux-thunk'
        import reducers from './reducers'
        const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ?
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({}) : compose;
        compose
```
调用方式：compose(...functions)
compose 用来实现从右到左来组合传入的多个函数，它做的只是让你不使用深度右括号的情况下来写深度嵌套的函数，仅此而已。
```
    const enhancer = composeEnhancers(applyMiddleware(thunk))
        const store = createStore(reducers, enhancer);
export default store;
```
```reducers.js
     import {combineReducers} from 'redux';
        import townReducer from './town';
        import communityReducer from './community';
        export default combineReducers({town: townReducer,community: communityReducer})
```
```` action.js
        import {GET_COMMUNITY_RANK_DATA,} from '../constant/community';
        function createValue(value, defaultValue) {return (value && value.result) ||
defaultValue}
        export const communityRankDataAction = (value) => ({
                type: GET_COMMUNITY_RANK_DATA,
                value: createValue(value, {})});

```` 模块化思维
constant 放置
````actionCreators      ----将actions和services联系触发 存redux
    import { communityRankStatusCreator, communityRankStatisticsCreator } from '../actions/town'
````

```
    import { queryCommunityRank, queryRatingStatistics} from '../../services/town'
    // 社区评级排名情况
        export function fetchCommunityRank() {
                return async (dispatch) => {
        const res = await queryCommunityRank()
        dispatch(communityRankStatusCreator(res))
                }
        }
    ....
·····页面调用 bindActionCreators
调用方式：bindActionCreators(actionCreators, dispatch)
通过connect的样式，将redux--state --映射到本页面
redux的 action ---映射一个将 actionCreators,dispatch 关联上的方式。
最后页面可以想正常调用函数调用此方法。
   Town.js --views
   function(props){
      useEffect(()=>{
          fetchCommunityRank();
      },[])
   }
const mapStateToProps = ({
  town: {
    communityRankStatus,
    communityRankStatistics,
  },
}) => {
  return {
    communityRankStatus,
    communityRankStatistics,
  };
};
const mapDispatchToProps = (dispatch) => {
  return bindActionCreators(
    {
      fetchCommunityRank,
      fetchRatingStatistics,
    },
    dispatch,
  );
};
export default connect(mapStateToProps, mapDispatchToProps)(Town);
```

你的本地仓库由 git 维护的三棵“树”组成。第一个是你的 工作目录，它持有实际文件；第二个是 缓存区（Index），它像个缓存区域，临时保存你的改动；最后是 HEAD，指向你最近一次提交后的结果

50.vue.3.0 的学习总结

vue.3.0 的学习总结

```
vue.3.0的优点：
更快，作者本人说新版的要比老版的快6倍多
① 从defineProperty到 proxy
```

a.他们完成数据双向的方式不同
b.代码更为简洁（不用for in循环）
```
  Object.defineProperty(ob,'a',{
      get: function(){},//依赖收集
      set: function(){},设置值，更新视图
  })
  ob.a    ob.a = 3;  (这里需要递归判断当前属性值是否为对象，对象就递归处理，否则就数据劫持.
var obj = {b: '4343', c:'4544"};
var newObj = new Proxy({
    obj, {
        get: function(target,key){return target[key]},
        set: function(target,key,newvalue){target[key] = newvalue; selft.render()},
    }
})
```
这里只是一个代理，所以还是要做一步：this.$data = new Proxy(this.$data,{...})
要代理的对象重新赋值。
newObj是返回了一个的对象，并不会污染原先对像，他只是个代理.
obj里面有多少个属性都会给你监听,不用在for in 循环了，效率更快
②重新定义vdom对比思路。

```
//virtual dom 是一个抽象层。
//他是为了方便为我们以js来描述html
//react -> 纯粹的js --提出虚拟dom方便描述这个html。
```

vue.3.0 的学习总结
创建项目 vue create myproject-vue3 --manany - balalba -- 3.0

1，向下兼容2.6x
3-1 reactive
*作用：创建响应式对象，非包装对象，可以认为是模板中的状态。
.tempalte 可以放兄弟节点
.reactive 类似useState，如果参数是字符串，数字，会报警告，value cannot be made reactive,所以应该设置对象，这样可以数据驱动页面。
3-2 ref
作用：创建一个包装式对象，含有一个响应式属性value。它和reactive的差别，就是前者灭有包装属性value
const count = ref(0),可以接收普通数据类型，count.value++
3-2-3 ref 访问dom或者组件
html ```` <input type="text" ref="myrefTxt" />
```
import{ref } from 'vue';
setup(){
    const myrefTxt = ref('');
    btn -- 调用  myrefTxt.value ---(dom)
}
```
ref和reactive的区别
ref不是响应式对象ref.value才是响应式对象，ref可以做数据拦截而react并不可以，
reactive是响应式对象.
同： ref和reactive都可以用于状态的管理。

3-2-2 toRefs
默认直接展开state,那么此时reactive 数据变成普通数据，通过toRefs，可以把reactive里的每个属性，转化为ref对象，这样展开后，就会变成多个ref对象，依然具有响应式特性，

```
 函数抽出成 hook
 ····hooks/useCount.js
 import { ref } from 'vue'
function useCount () {
  const count = ref(0)
  const handleAdd = () => {
    count.value++
  }
  const handleIncrease = () => {
    count.value--
  }
  return {
    count,
    handleAdd,
    handleIncrease
  }
}
export { useCount }
<template>
  <div>
    {{ state.name }}-{{ state.age }}-{{ state.count }}
    <button @click="handleIncrease">-</button>
    {{ state.count }}
    <button @click="handleAdd">+</button>
  </div>
</template>

<script>
import { reactive } from 'vue'
import { useCount } from './hooks/useCount'
export default {
  setup () {
    const { count, handleAdd, handleIncrease } = useCount()
    const state = reactive({
      name: 'guozhou',
      age: 18,
      count: count    //此处合理的调用了自定义hook函数又结合自身的响应数据
    })
    return {
      // count: state.count,
      state,
      handleAdd,
      handleIncrease
    }
  }
}
</script>
------------------hooks抽成的函数完全可以独立，工厂函数，他用了return 新对象的方法。
保持了独立性。
```

3-2-2toRefs
默认直接展开state,那么此时reactive数据变成普通，通过toRefs,可以把reactive里的每个属性，转化为ref对象，这样展开后，就会变成多个ref对象，依然具有响应式特性。
把页面调用的 state.name 改成 name调用 并响应式

```
setup () {
    return {
        //state,
        ...toRefs(state),
    }
}
```

父子通信

````
····parent
<child myTitle = "shouye" />
``````child
html
   {{mytext}}
export default {
    props:['myTitle'],
    setup (props) {
        const showTitle = props.myTitle + '1111';
        const mytext = ref(showTitle);
        return {
            mytext
        }
    }
````

vue3.0的生命周期使用

````
vue3.0的生命周期使用
vue2.6 ---- vue3的生命周期对比
原方法             升级后
beforeCreate      setup
created           setup
beforeMount       onBeforeMount
mounted           onMounted
beforeUpdate      onBeforeUpdate
updated           onUpdated
beforeDestroy     onBeforeUnmount
destroy           onUnmounted
import {onMounted,onBeforeUnmount,onUnmounted,onUpdated,onBeforeUpdate,ref} from 'vue'
```js
        setup () {
    // 状态,
    // 注册回调
    onMounted(() => {
      console.log('onMounted')
    })
    onBeforeUpdate(() => {
      console.log('onBeforeUpdate')
    })
````

```
    onUpdated(() => {
      console.log('onUpdated')
    })
    onBeforeUnmount(() => {
      console.log('onBeforeUnmount')
    })
    onUnmounted(() => {
      console.log('onUnmounted')
    })
    const myname = ref('chenguozhou')
    const handleClick = () => {
      myname.value = '乘过舟'
    }
    return {
      myname,
      handleClick
    }
  }
}
```

```
3-4 计算属性
computed(回调函数)
   setup() {
      const myText = ref('')
    const list = ref([])
    const computedList = computed(() =>{
      return list.value.filter(item => {
        return                                                    item.includes(myText.value)
      })
    })
   }
```

```
3-5 watch
监听器 watch是一个方法 ，它包含两个参数
 第一个参数是监听的值，count.value表示 当count.value发送变化就会触发监听器的回调函数，即第二个参数，第二个参数可以执行监听时候的回调。
 const reactivedata = reative({count: 1})
 const text = ref('');
 watch(()=>{
     reactivedata.count,
     val => {
         console.log('count is ${val}')
     }
 })
 watch(text,
  val => {
     console.log('count' is ${val})
  }
 )
 ref的数据监听模式
 const myText = ref('')
 watch(myText,()=>{
```

```
    console.log('666',myText);
})
reactive的数据监听模式
const state = reactive({
    myname: 'guozhou'
})
watch(()=>state.myname,()=>{
    console.log('myname',myname);
})
```

```
4-1路由
import{createRouter,createWebHistory,createWebHashHistory} from 'vue-router'
4-2 获取$router
···html
 @click= handleClick(item)
 ``` js    ctx.$router.push （不推荐这种方法）
  里面无法获取 整改item
 传item.id 就可以获取 id
import { getCurrentInstance } from 'vue'
setup () {
    const { ctx } = getCurrentInstance()
    //编程式导航
    ctx.$router.push('/home');
   注意点 这个实例执行要在setup周期函数里面最外层，不能再其他执行函数里面获取。
}
推荐方法二:
const router = useRouter() vue-router中useRouter直接获取router对象
    ctx.$router == this.$router(之前写法)

import {useRoute,useRouter} from 'vue-router'
$router = useRouter()
$route = useRoute()
$router.push('/detail/id');
const id = $route.params
```

```
vuex
拿到 页面拿到store的两种方法
①const store = useStore();
const {ctx} = getCurrentInstance();
const storeCount = computed(()=>{
    return ctx.$store.state.count
})
add(){
    ctx.$store.commit('addMutation')
}
```
```

```
var n = {
   toString: function () {
       return 1
   },
   valueOf: function () {
      return 2
   }
}
var obj = {1: 1, 2: 2};
console.log(+n);//2   //执行运算就是调用了valueOf方法
console.log(obj[n]);//1 // 获取值，调用了toString方法
```

```
webpack的面试点
module，chunk 和 bundle三者的区别。

module，chunk 和 bundle 其实就是同一份逻辑代码在不同转换场景下的取了三个名字：

我们直接写出来的是 module，webpack 处理时是 chunk，最后生成浏览器可以直接运行的 bundle。

避免缓存两个配置
①可以用chunkhash,而不是用hash，直接用hash会导致第三方库如vue，被打包。
（改了app.js，第三方库就不会动)--只更改业务代码.

②用webpack.HashedModuleldsPlugin-保持moudle.id稳定（如删除 import vueRouter from 'vue-router';不会导致 vue-router包减少。）
webpack.DefinePlugin -再打包阶段定义全局变量。
webpack.NoEmitOnErrosPlugin-屏蔽错误（打包过程遇到问题，跳过问题继续打包）
webpack.ProvidePlugin-提供库
（定义一些全局插件）
e.g:
  new webpack.ProvidePlugin({
      "$": jquery,
      "$axios": axios,
  })
copy-webpack-plugin-可以帮助拷贝内容（比如暂时不用的图片后面可能会用到，就可以用这个先拷贝到指定的目录）
new CopyWebpackPlugin([
    {
     from: path.resolve(__dirname,'../static'),
     to:config.build.assetsSubDirectory,
     ignore:['.*']
    }
])
//webpack 只会把他处理的资源放到dist里面
优化的内容
  Dll优化：打包速度的优化
  速度从 1846ms减少到为499ms
  dll的核心原理和概念：
    改代码-业务代码（不会动第三方模块）
    webpack.dll.js
    ...
    const webpack = require('webpack);
    module.exports ={
```

```
        entry: {
            vendor:['jquery','loadsh]
        },
        output: {
            path:__dirname+'dll',
            filename:"[name].dll.js",
            library:'[name]_library'
        },
        plugins:[
            new webpack.DllPlugin({
                path:—dirname+"/dll/[name]-manifest.json, //这个用于通知本地已经有这些数据了,
                // 这些库，不经常升级的vue,react,jquery库，下次就不会打包了,
                // 本地 html，script就会引入，以后页面调用vue或者react就会优先用打包的.
                name:'[name]_library'
            })
            // 生成一个json，让他不要再打包,(output的 library 命名和这里的名字name要同一个)
        ]
    }
```

```
webpack.config.js
        const webpack = require('webpack');
        module.exports = {
        mode: 'development',
        entry: {
            app: './app.js'
        },
        output: {
            filename: '[name].js',
        },
        module: {
                rules:[{
                test: /.js$/,
                use: [{loader:'babel-loader}
                }
            ]
        }]
    }
  }
```

npm install happypack -S Happypack ---处理少文件反而时间更多，文件多的时候，是得到了优化的 可以打包js也可以css new HappyPack({ id: "happybabel", loaders:['babel-loader?cacheDirectory=true'], threadPool: happyThreadPool }), new HappyPack({ id: "happycss", loaders:['css-loader?cacheDirectory=true'], threadPool: happyThreadPool }) 解决方案归纳 如果是要对模块内容进行处理 1.Loader是第一解决方案 如果要增加一些特殊的功能 1.可以自定义增加插件 项目上的打包简化，可变性配置等 1.通过编写相应的操作函数

```
##  51.父子组件的顺序

````
执行顺序 home-beforeRouteEnter --home-beforeCreate   -- child- beforeCreate --
--home-created --child-created --home-beforeMounted -- child -beforeMounted --
--child-mounted --home-mounted --home-beforeRouteLeave --home-beforDestoryed --
```
```

--child-beforeDestoryed -- child-Destoryed --home-Destoryed
````

## 52.异步组件，组件懒加载3种方法

````
①：结合Vue异步组件与webpack代码
const index = r=> require.ensure([],()=>r(require('@/page/index')),'index')
②： AMD写法
const Home = resolve => require(['../components/page/Home.vue'],resolve);
③：ES6写法
const Chat = ()=>import('@/components/page/Chat.vue')
配置完后需要再new Router 中使用。
整个项目运用到webpack的代码分割（webpack特性require.ensure()
1.vue的异步组件，2webpack的代码分割
````

53.cmd和amd的区别

```
cmd：依赖就近
--调用才加载
   define(function(requre,exports,module)       {
       var a = require('./a');
       a.doSomthing();
   })
amd：依赖前置 （定义模块和加载模块，而require只能加载模块）
---异步加载
   defined(['./a','./b'],function(a,b){
       a.doSomething();
       b.doSomething();
   })
```

前端统一，nodejs环境中，遵循commonJs规范来组织模块
在浏览器环境中，遵循ES Modules规范。（ECMAScript 2015(ES6)）

webpack 空杯思想重学
vue cli create-react-app 高度集成的cli，面对特殊资源加载，打包过程优化、资源代码分块、Tree-shaking这样相对复杂的需求无从下手。
webpack的高级特性： Source Map、模块热替换（HMR)机制、Proxy、webpack Dev Server 等周边技能的使用以及Tree-Shaking、sideEffects、Code Spliting 等高级特性的实践。
再有就是常用优化插件、三种hash的最佳实践，打包速度优化
其他同类优秀方案： Rollup、Parcel
让你能够了解到一些webpack同类的优秀方案，以及他们设计上的不同。
1.
webpack将散落的模块打包到一起。（散落的javascript 打包到 一个 js文件中);
支持不同种类的前端模块。
2.
对于有环境兼容问题的代码
webpack 可以在打包过程中通过loader 机制对其实现编译转换，然后再进行打包。
3.
对于不同类型的前端模块，webpack支持在javaScript中以模块化的方式载入任意类型的资源文件。
4.
具备代码拆分的能力
```

能够将应用中所有的模块按需分块打包。--避免全部代码打包到一起，产生单文件过大，导致加载慢的问题。
demo
npm init --yes npm i webpack webpack-cli --save-dev (webpack-cli是cli程序让webpack可以在命令行中调用)
npx webpack --version
        webpack 5.11.1
        webpack-cli 4.3.0
        webpack-dev-server 3.11.0
//注释调这段辅助代码，写给vscode的看的 import {Configuration} from 'webpack';这样webpack配置会有提示减少出错
// import {Configuration} from 'webpack';

```
/**
 * @type {configuration}
 * */

const config = {
  entry: './src.index.js',
  output: {
    filename: 'bundle.js',
    path: __dirname + '/dist'
  },

}
module.exports = config;
```

5.
webpack的工作模式
webpack针对不同环境的三组预设配置：
*production 模式下
启动内置优化插件，自动优化打包结果，打包速度偏慢。
*development 模式下
自动优化打包速度，添加一些调试过程中的辅助插件以便于更好的调试错误。
*none模式下
运行最原始的打包，不做任何额外处理，这种模式一般需要分析我们模块的打包结果时会用到
6 webpack 工作模式
想要修改webpack工作模式的方式有两种：
*通过cli--mode参数传入
*通过配置文件设置mode属性
tips: -vscode 折叠代码的快捷键 Ctrl+K, Ctrl+0(masOs: Command+0)
bundle.js   cli   serve . 启动一个http 服务

501面试题

面试题
1.vuex是双项据流吗?
2.前端工程化，有哪些?
3.webpack是什么工具?构建工具，打包工具? 基础配置
本质：webpack是模块化打包工具。万物皆是模块。
4.vue.2 arr[3]=4，页面会重新渲染吗

5.什么是纯函数，array的splice是不是 纯函数？纯函数分为什么？
6.缓存，跨域，es6，
7.通信方式vue
8.组件拓展方式
9.项目优化
11.this指向问题。
12.事件循环，函数式编程，宏任务和微任务的理解
13.版本回退问题，git如何管理项目
探迹科技
1. MVC,MVP,MVVM的差异
2. MVC是在哪里做逻辑处理的
3. Object.defineproty和Proxy的区别
4. 使用Object.defineproty怎么监听数组的变化
5. 不用keep-alive，需要缓存2000个组件，超过后将最久没使用的组件回收处理
6. A组件引用B组件，B组件引用A组件，会有什么问题
7. vue组件递归有什么问题
8. promise为什么可以一直then
9. 怎么做回调优化
10. promise的then不断返回新的promise，怎么抹平其中的差异
11. 除了seo这个原因之外，为什么要用ssr
视源股份
1.jsx是什么，template模板？
2.为什么虚拟dom可以优化性能
3.节点对比，怎么对比，怎么更新，什么api，如何比较，下标如何移动？
4.vue和react的什么周期，优化性能reacthook
南方电网
1.防抖节流分别是什么，具体代表？
2.git的工作流，如何合并一天的git提交记录，版本冲突回退问题。
3.cmd,amd规范
完美日记
1.前端模块化，前端工程化？
统一的开发
标准化，是一种思想
2.讲一下
3.虚拟dom,diff算法，节点对比，怎么对比，怎么更新，下标如何移动。
4.vue和react的生命周期，优化性能，reactHook
5.webpack的module是做什么的？
6.防抖节流
7，浅克隆和深克隆
8.时间复杂度和空间复杂度
9.异步同步原理
10.原型链和rem原理
北明软件
1.异步同步原理，vue原理，reacthook,高级特性，ts,从0搭建webapck;
同步：是按照任务循序依此执行，当前的这件事没有执行完，下一件事情会阻塞执行不了。
异步，当前这件事情没有执行完，需要等待一段时间才能继续处理，此时我们先不等，继续执行下面的任务，
当后面的任务执行完成了，去吧没有彻底执行完的任务，完成了。
浏览器是多线程的，浏览器只分配一个线程给js，
2.事件循环机制（浏览器原理）
浏览器是多线程的，浏览器只分配一个线程给js，
这个主线程上执行形成一个执行栈，和任务队列，和web API(异步进程) 和事件循环。
①主线程读取js代码，为同步环境，读取当前代码console.log，入栈，打印出东西，出栈，
遇到异步任务setTimeout(fn,2000),读取setTimeout,将 延迟2秒的fn推给异步进程 webApis，

然后seTimeout，出栈执行结束，
如果此时主线程没有任务执行了，等待2秒，到达条件异步进程将函数fn推入任务队列中，按照顺序事件依次退出任务队列，
主线程无任务会调用事件循环，查询到当前任务，推出第一个任务，到主线程，
然后继续读取fn，入栈，执行fn,后出栈，如果当前主线程又没有任务就会调用事件循环，查询任务队列，将第一个推出到主线程执行。
如此重复执行
面试题
1。fetch和ajax的区别是啥?
2.场景题，假设你用手机做题，做到一半手机没电了，你用新的手机继续做题要保持进度，要怎么解决
3.react,vue.2x和vue3.0的diff算法分别是怎么样的？有什么区别
4.项目中webpack用过吗？那么cdn如何引入在webpack项目中
5.webpack的plugin和loader的区别，分别是干嘛的？具体讲讲，loader里面有各个转化箭头函数的loader，怎么配置然后这个loader内部他是怎么转化的。
6.vue中的观察者模式?
7.es6中的原型继承
8.循环的区别
9.数据类型
10隐式转换
11.es6中的Set和Map的区别
Set 对象类似于数组，且成员的值都是唯一的，
Map 对象是键值对集合，和 JSON 对象类似，但是 key 不仅可以是字符串还可以是对象
12.es6如何实现异步如：一个列表很长，如何先渲染前面部分
13.null和undefined的区别
14.react的性能优化
15.深比较和浅比较，是什么，浅比较，和深比较比较到哪层
16.react的purecomponent是干嘛，讲讲?
17.如何实现或者监听数组索引?

```
//function updateView(){console.log('视图更新')};
//重新定义数组原型
const oldArrayProperty = array.prototype;
//创建新对象，原型指向oldArrayProperty,再拓展新的方法不会影响原型
const arrProto = Object.create(oldArrayProperty)
['push','pop','shift','unshift','splice'].forEach(methodName=>{
    arrProto[methodName] = function(){
        updateView() //触发视图更新
        oldArrayProperty[methodName].call)(this,...arguments)
    }
})
//重新定义属性，监听起来
function defineReactive(target,key,value){
    observer(value)
}
```

18.computed原理?
19.class的super做了什么,class有原型对象吗?
class是有原型对象的protype.静态属性和静态方法，前加了Static，
继续的实例拿不到这个属性和方法，
super，类私于call，在函数继承的时候，call他的属性和方法过来，super就是拿到父类的属性和方法
20.vue和react的生命周期区别?
一个React组件的生命周期分为三个部分：挂载期（Mounting）、存在更新期（Updating）和销毁时（Unmounting）。
mounting：
    constructor,初始化继承react的props,state初始化

componentwillMount 类似于vue的beforeMount
render 类似于 vue的template,该方法创建了virtualDom
componentDidMount 类似于 vue的 mounted;
Updating
通过改变props和state来驱动视图的更改，触发以下钩子
componentWillReceiveProps (props-change触发)
shouldComponentUpdate 性能优化点 (props-state两者改变都会触发)
componentWillUpdate 类似于beforeUpdate
render()
componentDidUpdate vue 的updated
Unmounting
componentWillUnmounting 类似于react的 destoryed

21.304,如何http进行缓存，
22.闭包原理，外层函数赋值为null会怎么样?
函数嵌套函数，外部函数的变量被内部函数所占用，最终会返回return 出这个内部函数
其实就是可以定义一个私有变量，私有变量。
如果已经调用了函数赋值给一个变量，那么没有影响，如果调用前将这个函数赋值，则无法执行
23.vuex的向数据流 ： ui-用store的state-需要改变，调用vuex的方法，dispatch(action),commit(mutation),改变数据，最后视图view，数据更新，响应更新。
24.手机兼容问题?
① uniapp的tabbar,底部导航，在搜索框输入文字弹出软键盘，被顶起问题--自己写导航，minxin
② 表单输入，容器没有设定最小高度，表单下面还有其他东西，这些键盘弹起，底下会有个空白。--设置最小高度
③ 苹果手机日期组件，在苹果手机出现的兼容问题.
④ uni.navigateTo({}),跳转，返回用uni.navigateBack ({delta:3}) ，这样不会出现问题
onBackPress(event){uni.redirectTo({url:'../index/index'});return true;}，监听左上角按钮事件
项目难点，用debug解决
25.南方航空官网项目讲解，主要负责海外站点购票流这一块，澳大利亚，加拿大，等占地，
①航班查询，选择去和返回目的地，还是单行程旅途，是日期控件，选择时间和地点，
②然后到了shop页，主要是航班展示和低价日历，选择具体的时间行程，预定
③然后到了旅客信息页，航程信息然后底下进行旅客信息的表单填写（基本信息和证件）和联系人信息
和优惠券（优惠券--输入优惠券密码），乘客选座（弹窗展示座位点击就可以选择）
④确认订单页，前面信息的一个展示，支付按钮，跳转支付页面（这个后端控制的）支付成功后弹窗展示跳到支付完成页面
⑤支付完成页面，航班信息展示，表格信息展示，底下有个引导快速入会按钮，跳转到快速入会页面，也是表单填写页面注册成为会员（会涉及表单信息回填）；
26.浩鲸项目讲解，浩鲸这边做了一个云展会，主要用了uniapp，搭建项目，webview内嵌h5的形式，
注册登录页面，前端croptyjs加密和图片验证，封装公共请求，（uni.request）定义一个class ideerRequest
定义公共请求方法，request(){return new Promise((resolve,reject)=>{
uni.request({...option},success(res)=>{resolve()},error:(err)=>{reject(err)})
})}
再分别定义get,post,delete,put方法请求 ： get(url,data){
const options = {
url:`${this.baseUrl}${url}`
data,
methods:'GET',
header:{
Accept: 'application/json',
'X-Access-Token': uni.getStorageSync('userInfo').token,
}
}
return this.request(options)

}
图片组件封装，用递归上传的形式，一个个上传，公共按钮，公共卡片，组件复用

27.webpack的优化，配置：
①首先我们项目呢是由，css,jpg,jsx或者js,vue,组成的，webpack，有个配置文件，webpack.config.js,他定义了一个入口文件，entry，从入口文件import的模块，找到对应的模块，形成一个AsT依赖树（抽象语法树），通过递归可以找到所有依赖的文件，然后通过交给 loader里的css-loader,less-loader,file-loader,babel-loader的各种loader转化各种类型的（css,图片，jpg，箭头函数)转化成浏览器能识别的东西。，然后通过plugin,加载各种资源，从打包优化和压缩,定义环境变量，处理各种各样的任务如htmlWebpackplugin，
定义打包成的bundlejs输出注入到的模板html文件，
当然webpack还有cleanWebpackplugin,清楚因每次打包生产不同文件名的缓存。
webpack，还可以定义devServer在开发的时候定义代理，请求访问后端
②优化：webpack的优化：
a.自身的优化 treeShaking 打包不会打包没有用到的文件
scope-hoting 作用域提升比如 console.log(a+b),最终打印3，不会吧ab变量打包进入
b 非自身的优化，
体积上的优化 如用moment是，语言包，不用到，可以用ignorePlugin,打包时就不会加多余的语言包打包进入
代码的优雅性 loader，用less-loader,css-loader postCss-loader,优雅写法和兼容css,添加-webkit-前缀
注入环境变量，DefinePlugin
重复打包框架，noParse 如react.min.js module={
noParse(/react\.min\.js/,react)
}
动态链接库 Dllplugin
等等，其他有需要可以查文档
28，项目中的优化，
首先，代码规范：项目搭建，引入eslint规范代码，
②css,用less，scss预处理器,更为优雅，最好不能超过三层
③背景图片用雪碧图，减少图片请求，
④封装公共请求，利用本地缓存，减少请求，封装复用组件，公共函数，减少编写多余的代码，尽量代码精简性
⑤按需引入，项目中引入ui框架按需引入，vue项目的化路由懒加载，reat，用lazy,subspend,切片js加载，
⑥如果不考虑seo，可以用ssr渲染，
⑦引入cdn，
⑧seo优化，img，图片alt,标注图片是干嘛的，使用语义化标签如header nav main footer等
⑨vue项目遍历渲染加key,对于多次切换的tab，用v-show，避免dom的多次销毁重建v-if
⑩项目本身就有压缩代码js,css,提取css，miniCssExtactplugin.loder
plugins: [
new MiniCssExtractPlugin({
filename: "css/common.css"
}),
]
29.react的高级特性hook useState ,useReducer(fn,state),useEffect(fn,[非要引用数据，造成死循环])
只能dom更新后-同compoentDidMount,可以用 useMemo(fn,[]) useMemo(fn) === useCallBack(fn)
30.浅层对比 pureComponent(props,state,变化) --- memo (props，return true阻止渲染) --
shouldComponentUpdate(return false ---阻止渲染)
31.vue.3 beforeCreate Created === setUp
const state = reactive({age:18}); return state === ...toRef(state)
32.面试官，还有什么问题要问的
问公司的开发流程时怎么样的，有没有预发布环境（数据真实，几个月前的）
测试环境数据自己造的，
真实环境时实时的。
有没有自动化部署
33.浏览器发起跨域请求，由于安全原因，浏览器自动发起OPTIONS请求，即为预检请求，CORS请求，如果服务接受请

求，浏览器才会发起正式请求
34.如何实现一个深度克隆一个对象?

```
function deepClone(obj){
var newObj= obj instanceof Array ? []:{};
for(var item in obj){
var temple= typeof obj[item] == 'object' ? deepClone(obj[item]):obj[item];
newObj[item] = temple;
}
return newObj;
}
```

51.微信小程序的学习

一.小程序性能优化的具体维度:
01.避免过大的wxml节点数码
　　wxml是基于html的一种DSL，常规组件最终会被小程序的渲染线程通过WebView渲染为HTML，大部分性能优化方案均适用于WXML。
02.避免执行脚本的耗时过长
　　在小程序加载完成后首次渲染期间
　　小程序运行过程中处理用户交互时期
03.避免首屏时间太长
　　影响首屏时间最核心的两个优化方向
　　*代码优化
　　*网络优化
　　渲染之前展示友好的loading效果。
　　降低WXML的结果复杂度
　　降低首次渲染的数据规模
　　网络方向的优化核心是为了降低RTT
　　a.最直观的网络优化方案--减少网络请求所携带的数据体积
　　b.对服务端的要求--提高服务器处理网络请求的速度
04.避免渲染界面的耗时过长
　　动态渲染是由JavaScript脚本中调用setData更新数据所触发，优化setData是优化动态渲染的切入点
05.对网络请求做必要缓存以避免多余的请求
06.所有请求耗时不应太久
07.避免setData的调用过于频繁
08.避免setData的数据过大
09.避免短时间内发起太多的图片请求
　　一个TCP链接同时只能处理一个HTTP，同一时间的HTTP请求太多就会产生排队
　　浏览器为应对问题建立多个TCP链接以实现并行发送HTTP请求的目的
10.避免短时间内发起太多的请求。
二.小程序工程化
1.善用webpack优化研发模式
　a.管理第三方npm模块
　　微信小程序再基础库2.2.1版本才开始支持使用第三方npm包
　　原始npm模块---node_modules
　　微信IDE构建npm
　　合成 miniprogram_npm （小程序可用npm模块）
　b.使用TypeScript编写源码
　c.使用预处理器编写模块化的样式
　d.使用lint工具统一源码规范
　e.图片压缩

f.多环境支持
2.测试金字塔模型
前端后端单元测试--集成测试---端到端测试---验收测试

52.promise的理解

一。源码
```
function Promise(executor) {
  var self = this
  self.status = 'pending' // Promise当前的状态
  self.data = undefined  // Promise的值
  self.onResolvedCallback = [] // Promise resolve时的回调函数集，因为在Promise结束之前有可能有多个回调添加到它上面
  self.onRejectedCallback = [] // Promise reject时的回调函数集，因为在Promise结束之前有可能有多个回调添加到它上面
  function resolve(value) {
    // TODO
  }
  function reject(reason) {
    // TODO
  }
  try { // 考虑到执行executor的过程中有可能出错，所以我们用try/catch块给包起来，并且在出错后以catch到的值reject掉这个Promise
    executor(resolve.bind(this), reject.bind(this)) // 执行executor
  } catch(e) {
    reject.bind(this)(e)
  }
}
```
二。外部调用,promise就是传入一个回调函数，promise源码接收传入excutor函数调用源码本身实参resolve和reject两个函数，这时就是会调用我们传入函数的参数cb1或者cb2。
```
//其他的内部执行就是一些状态改变和将传入的方法存储在调用而已.
    const excutor = (cb1,cb2)=>{
        uni.request({
           ...options,
           success:(res)=>{
               cb1(res)//cb1就是源码本身函数resolve(res),调用源码成功函数
               cb2(res)//cb2就是源码本身函数reject(res)，调用源码失败函数
           }
        })
        }
        var myPromise = new Promise(executor)
```
三。源码内部的 resolve,reject做了啥
```
function resolve(){
    if(self.status === 'pending'){
        self.status = 'resolved'
        self.data = value    for(vari=0;i<self.onResolvedCallback.length.i++){
     self.onResolvedCallback[i](value);
    }
}
function reject(reason){
    if(self.status==='pending'){
        self.status = 'reject';
```

```
        self.data = reason;
        for(var i=0;i<self.onRejectCallback.length;i++){
        self.onRejectedCallback[i](reason);
    }
}
```
```then方法的实现
Promise.prototype.then= (obResolved,onRejected)=>{
    var self = this;var promise2;
    onResolved = typeof onResolved === 'function'? onResolved: function(v) {
        return v
    }
    onRejected = typeof onReject === 'function' : function(r) {
        throw r
    }
    //根据标准，如果then的参数不是function,则我们需要忽略他，此处以如下方式处理：
    if(self.status === 'resolved'){
        setTimeout(function(){//异步执行onResolved
            try {
                var x = onResolved(self.data)
                resolvePromise(promise2,resolve,reject)
            } catch(reason){
                reject(reason)
            }
        })
    }
    if(self.status === 'rejected'){
        return promise2 = new Promise(function(resolve,reject) {
            setTimeout(function(){
                try{
                    var x = onResolved(self.data)
                    resolvePromise(promise2,x,resolve,reject)
                }catch(reason){
                    reject(reason)
                }
            })
        })
    }
    //////
    if(self.status === 'pending'){
        return promise2 = new Promise(function(resolve,reject){
            self.onResolvedCallback.push(function(value){
                try{
                    var x = onResolved(value);
                    resolvePromise(promsie2,x,resolve,reject)
                }catch(r){
                    reject(r)
                }
            })
            self.onRejectedCallback.push(function(reason){
                try{
                    var x = onRejected(reason);
                    resolvePromise(promise2,x,resolve,reject)
                } catch(r){
```

```
                    reject(r)
                }
            })
        })
    }
}
//////////////////////
Promise.prototype.catch = function(onRejected){
    return this.then(null,onRejected)
}
Promise.deferred = Promise.defer = funciton(onRejected){
        var dfd = {};
        dfd.promise = new Promise(function(resolve,reject){
        dfd.resolve = resolve
        dfd.reject = reject
        })
        return dfd;
}
```

53.diff算法 react,vue2.x vue3.0的对比

文档来源：https://juejin.cn/post/6919376064833667080

```
mount(vnode, parent, [refNode]):  通过vnode生成真实的DOM节点。parent为其父级的真实DOM节点，refNode为真实的DOM节点，其父级节点为parent。如果refNode不为空，vnode生成的DOM节点就会插入到refNode之前；如果refNode为空，那么vnode生成的DOM节点就作为最后一个子节点插入到parent中

patch(prevNode, nextNode, parent): 可以简单的理解为给当前DOM节点进行更新，并且调用diff算法对比自身的子节点;

A.移动节点
react的diff，按照顺序 preChildren a-b-c
nextChildren c-a-b
更新是 a往c移动，b晚a挪动。
先遍历最新的nextChildren的每一项元素，找出再旧节点的位置，记录当前的最后一个相同不需要改变的位置lastIndex=3(当前nextChildren的索引位置)，
最后找出现在有变化的项在旧的节点位置为1，
preChildIdx < lastIndex;
所以要移动。
使用patch（更新节点，到vnode.el)
B.增加节点
循环新vnode元素，定义一个变量有key同就是为true，无默认false，
最后当层循环结束，
if(!find){//插入节点
    let refNode = i<=0?
    prevChildren[0].el :
    nextChildren[i-1].el.nextSibling
    mount(nextChild,parent,refNode);
}
C.移除节点
旧列表存在，新列表不存在。
整个两层循环结束，
```

再遍历旧列表的key,在新列表过滤出存在的数组，若数组为空，挪出当前元素。
优化：不足点：
目前的reactDiff的时间复杂度为O(m*n)，我们可以用空间换时间，把key与index的关系维护成一个Map，从而将时间复杂度降低为O(n)

vue.2.0 diff--双端比较 解决了此问题
（所谓双端比较就是新列表和旧列表两个列表的头与尾互相对比，，在对比的过程中指针会逐渐向内靠拢，直到某一个列表的节点全部遍历过，对比停止。）

使用旧列表的头一个节点oldStartNode与新列表的头一个节点newStartNode对比
使用旧列表的最后一个节点oldEndNode与新列表的最后一个节点newEndNode对比
使用旧列表的头一个节点oldStartNode与新列表的最后一个节点newEndNode对比
使用旧列表的最后一个节点oldEndNode与新列表的头一个节点newStartNode对比

当对比时找到了可复用的节点，我们还是先patch给元素打补丁，然后将指针进行前/后移一位指针。根据对比节点的不同，我们移动的指针和方向也不同，具体规则如下：

当旧列表的头一个节点oldStartNode与新列表的头一个节点newStartNode对比时key相同。那么旧列表的头指针oldStartIndex与新列表的头指针newStartIndex同时向后移动一位。
当旧列表的最后一个节点oldEndNode与新列表的最后一个节点newEndNode对比时key相同。那么旧列表的尾指针oldEndIndex与新列表的尾指针newEndIndex同时向前移动一位。
当旧列表的头一个节点oldStartNode与新列表的最后一个节点newEndNode对比时key相同。那么旧列表的头指针oldStartIndex向后移动一位；新列表的尾指针newEndIndex向前移动一位。
当旧列表的最后一个节点oldEndNode与新列表的头一个节点newStartNode对比时key相同。那么旧列表的尾指针oldEndIndex向前移动一位；新列表的头指针newStartIndex向后移动一位。

上面都是交叉对比能找到的，
还有一种特殊的情况：四次对比都没有找到的
⑤我们只能用新列表的第一节点去旧列表中找与其key相同的节点。
找的结果还分两种情况，a.旧列表中找到了，b.没有找到。

```
function vue2Diff(prevChildren, nextChildren, parent) {
  //...
  while (oldStartIndex <= oldEndIndex && newStartIndex <=newEndIndex) {
    if (oldStartNode === undefind) {
        oldStartNode = prevChildren[++oldStartIndex]
    } else if (oldEndNode === undefind) {
        oldEndNode = prevChildren[--oldEndIndex]
    } else if (oldStartNode.key === newStartNode.key) {
    //...
    } else if (oldEndNode.key === newEndNode.key) {
    //...
    } else if (oldStartNode.key === newEndNode.key) {
    //...
    } else if (oldEndNode.key === newStartNode.key) {
    //...
    } else {
    // ...
    }
  }
  //添加节点和移除节点
  if (oldEndIndex < oldStartIndex) {
    for (let i = newStartIndex; i <= newEndIndex; i++) {
      mount(nextChildren[i], parent, prevStartNode.el)
```

```
    }
  } else if (newEndIndex < newStartIndex) {
    for (let i = oldStartIndex; i <= oldEndIndex; i++) {
      if (prevChildren[i]) {
        partent.removeChild(prevChildren[i].el)
      }
    }
  }

}
```

三。vue3.0diff--最长递增子序列
vue3的diff借鉴于inferno，该算法其中有两个理念。第一个是相同的前置与后置元素的预处理；第二个则是最长递增子序列，此思想与React的diff类似又不尽相同。下面我们来一一介绍。

react的diff算法用了递增方法，两层循环，第一层最新虚拟节点遍历，第二层是旧的节点遍历，传进来的参数是newVnode,oldeVnode,parent,
分改，增，删，三种情况
改，取出最新从旧列表遍历对比相同key,如果旧key索引大于等于最新记录索引（0开始),更新最新lastindex,
反之，如果旧key小于lastIndex，移动位置（真实dom，当前最新Vnode列表,前一个节点对应真实dom，的后一个元素nextChildren[i-1].el.nextSibliing 空间复杂度为O(m*n) 【更新dom的方法前一个元素的下个兄弟元素更新）
vue2.x 双端对比，头尾互对比
两端向中间靠拢，新旧挪一位
patch函数对比Vnode和oldeVnode，如果不一样，更新真实dom，为Vnode--做了增删改，
createElement,removeChild,update
vue同层对比，时间复杂度为O（N)非常高效

54.fetch和ajax的区别

fetch是全局window的一个方法，
特点：①第一个参数是URL，
②第二个是可选参数，可以控制不同配置的init对象
③使用了javaScript Promise来处理结果/回调：

```
  fetch('/some/url')then(function(res){return .....执行成功第一步}).then(function(returnValue){
      //执行成功第二步
  })。catch(function(err){
      //中途任何地方出错。。。在此处理
  })
```

③fetch返回得promise将不会拒绝HTTP错误状态，即使响应是一个http404或者500.相反，它会正常解决（其中ok状态设置未false），只有在网络故障时或者任何阻止请求完成时，才会拒绝。
封装

```
  function checkStatus(response) {
  if (response.status >= 200 && response.status < 300) {
    return response
  } else {
    var error = new Error(response.statusText)
    error.response = response
    throw error
  }
}
```

```
function parseJSON(response) {
  return response.json()
}

fetch('/users')
  .then(checkStatus)
  .then(parseJSON)
  .then(function(data) {
    console.log('request succeeded with JSON response', data)
  }).catch(function(error) {
    console.log('request failed', error)
  })
  @默认情况，fetch在服务端不会发送和接受任何cookies，要发送必须
  fetch(url, {
  credentials: 'same-origin'
})
cors请求
fetch(url, {
  credentials: 'include'
})
```

55.安全性问题

三大安全问题：
一.xss攻击：
站内伪造类，利用表单内的脚本，转义，如果是表单内容为脚本的

二.CSRF攻击：跨站请求伪造
<meta name="referrer" content="origin">
三：点击劫持：
攻击者将需要攻击的网站通过 iframe 嵌套的方式嵌入自己的网页中，并将 iframe 设置为透明，在页面中透出一个按钮诱导用户点击。
方案一：
```
if (self !== top) {
// 跳回原页面
top.location = self.location;
}
```
方案二：使用 HTTP 中的 x-frame-options 头，控制 iframe 的加载，它有 3 个值可选：
{sandbox : 'allow-same-origin'//则只能加载与主站同域的资源
DENY，表示页面不允许通过 iframe 的方式展示
SAMEORIGIN，表示页面可以在相同域名下通过 iframe 的方式展示。
ALLOW-FROM，表示页面可以在指定来源的 iframe 中展示

防盗链，图片防盗：
前端设置
<meta name="referrer" content="origin">
请求头 Request Headers
Rerferer： http://localhost:3000/river/anomaly/index
火狐浏览器可以修改network.http.sendRefererHeader
（方法：about:config ---- 搜索network.http.sendRefererHeader）

```
wepack 配置 ngnix,代理抓包，重写路径
proxy: {
    '/api/':{
        bypass: function(req,res,proxyOptions){
            for(let i=0;i<proxyList.length;i++){
                const item = proxyList[i];
                if(req.url.includes(item.url)){
                    return item.file
                }
            }
        }
        return req.url;
    }
}
```

56.react Hooks

```
1.如何实现同vue的computed的功能
import React, { memo, useMemo, useCallback, useState } from 'react';

const App = memo(() => {
  const [count, setCount] = useState(0);
  let double = useMemo(() => {
    return count * 2
  }, [count]);   //double依赖于count，当count改变时，double自动改变，详情可见我的useMemo文章
  return (
    <div>
      <button onClick={()=>{setCount((count) => count + 1)}}>count+1</button>
    </div>
  )
})
export default App

useEffect(()=>{},[])
相当于componentDidMount，第二参数为数组，相当于设置了依赖，当这个数组的值变化，就会执行第一个参数的回调函数。
<div>
{price}
<p>{name}</p>
<p>{getProduceName()}
<button onClick={()=>setPrice(price+1)}></button>
</div>
缺点：目标是在DOM发生变化时，不想关的函数不需要触发，而effect只能在DOM更新后再触发
state,price-也就是dom里面的-div变化，才到useEffect
用useMemo,可以解决这个问题.
区别：
1.useEffect是在渲染之后完成的
2.useMemo是在渲染期间完成的  useMemo=(()=>{},[])第二个参数为空数组只执行一次
        useMemo( ()=>{fn} ) 等价于 useCallback(fn)
3.memo--hook，pureComponent-类组件
memo类似于PureCompoent 作用是优化组件性能，防止组件触发重渲染
memo针对 一个组件的渲染是否重复执行
```

57.继承的方式

```
1.原型链继承
 子类原型 = 父类实例
2.原型链继承+借用构造函数继承
 子类原型对象指向= 父类实例 ，父类属性需要就call过来，parent.call('属性1','属性2');
3.原型式继承
 子类原型对象指向 = （临时函数的原型对象指向父类的原型对象）实例化临时对象
4。寄生组合继承
子类原型对象指向 = 临时函数Object.create(父类实例)+借用构造函数。拿到父类属性.call('属性1','属性2')
```

```
1.vue的双向绑定原理，vue的响应原理?
vue通过设定对象属性的 setter/getter 方法来监听数据的变化，通过getter进行依赖收集，而每个setter方法就是一个观察者，在数据变更的时候通知订阅者更新视图。
3.react的子组件传值
4.vue模板挂载是怎样的?
5.做了哪些提高用户体验度的事情?
6.项目重构有哪些需要做的
7.项目难点
8.useEffect的第二个参数?
第二个参数是外部的数据变量，
useEffect的不作为componentDidmount的话，传入第二个参数时一定注意：第二个参数不能为引用类型，引用类型比较不出来数据的变化，会造成死循环
useEffect(()=>{},[count])
9.react的传值方式?
props,redux,refs,context ,hoc
React.createRef('xxx'),高阶组件 hoc   render props
```

58.全栈小程序开发学习

```
小程序可以做hybrid做不了的的事情，消息订阅和推送
一.小程序的运行机制
```

热启动	冷启动
已经打开过	用户首次打开
（一定时间）再次打开	（销毁关闭）再次打开
无需重新启动	重新加载启动