1. vue

1.1. 解释什么是生命周期

Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载 DOM->渲染、更新->渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。

1.2. 生命周期的作用是什么

它的生命周期中有多个事件钩子,让我们在控制整个 vue 实例的过程时更容易形成好的逻辑

1.3. 生命周期总共有几个阶段

常用的分为 4 个大的阶段：有创建,挂载，更新，销毁这 4 个阶段

1.4. 第一次页面加载会触发哪几个钩子

beforeCreate,created,beforeMount,mounted

1.5. DOM 渲染在哪个周期中就已经完成?

DOM 渲染在 mounted 中就已经完成了

1.6. created 中能操作 dom 吗? 非要操作怎么办?

不能直接操作，但是可以通过 nextTick 来进行操作

1.7. 生命周期钩子的一些使用方法

1.beforecreate:可以在加个 loading 事件，在加载实例是触发
2.created:初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用
3.mounted:挂载元素，获取到 dom 节点
4.updated:如果对数据统一处理，在这里写上相应函数
5.beforeDestroy:可以提供确认停止事件的确认框

1.8. v-show 与 v-if 的区别

v-show 是 css 切换，v-if 是完整的销毁和重新创建
v-show 仅仅控制元素的显示方式，将 display 属性在 block 和 none 来回切换；而 v-if 会控制这个 DOM 节点的存在与否。当我们需要经常切换某个元素的显示/隐藏时，使用 v-show 会更加节省性能上的开销；当只需要一次显示或隐藏时，使用 v-if 更加合理。

1.9. v-if 与 v-for 的优先级

官方不建议我们同时使用。

在 vue2，v-for 的优先级会高于 v-if。
3” v-for=”item in [5,1,4]”>{{item}} 出现 5,4

在 vue3 中，v-if 的优先级会高于 v-for
3” v-for=”item in [5,1,4]”> 什么都不渲染

1.10. key 的作用是什么

主要用来标记节点，为虚拟 dom 的计算提供数据。

1.11. 什么是虚拟 dom

虚拟 dom 本质上是一个对象，封装了对应真实 dom 的相关属性信息，以及框架设定的一些属性信息(@click)

他的作用：

当数据发生改变的时候，优先操作虚拟 dom，将新的虚拟 dom 和之前的虚拟 dom 使用 diff 算法，找出变化的部分，将变化的部分编译成真实 dom，渲染到页面上，尽可能减少页面重排与重绘，提高性能。

1.12. vue 组件中的 key 有什么用

key 的作用就是更新组件时判断两个节点是否相同。相同就复用，否则就删除旧的创建新的

1.13. key 的值用 index 比较好，还是用属性比较好

用属性会更好一些

<div v-for((item,index) in [{id:1},{id:3},{id:4}]) :key=”index”>

1.14. v-model 的作用

v-model 是用来作为模型和视图的双向绑定的。负责将页面的数据，同步绑定到 vue 实例当中，将 vue 实例中的数据，实时渲染到页面上去。

1.15. 以及双向绑定的原理

v2

vue.js 是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，触发相应的监听回调来渲染视图。

v3

vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过 Proxy 代理来劫持各个属性的 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来渲染视图。

1.16. 什么是数据响应式

数据响应式是指页面能够响应数据的变化，或者数据能够响应页面的变化，都可以称之为响应式。两个变化合并在一起，形成了双向绑定。

1.17. v-text,v-html,插值表达式的区别

插值表达式，里面不能含有节点，可能有闪烁现象，不覆盖元素的内容

v-text，不会有闪烁现象，不能解析节点

v-html，既不会有闪烁现象，也能够解析节点

1.18. v-on

用来绑定事件的。
<button v-on:click="alert(33)"><button>
也可以简写
<button @click="alert(33)"><button>

1.19. v-bind

用来绑定属性的值。

可以简写

1.20. 绑定 class 的数组用法

对象方法:class="{'orange':true, 'green':false}"
行内　　:style="{color:'red',fontSize:'12px'}"

1.21. method,computed 和 watch 有什么区别

method
method 表示方法，不具有缓存性，返回值可有可无

computed
computed 是计算属性，也就是计算值，它更多用于计算值的场景
computed 具有缓存性，computed 的值在 get 执行后是会缓存的，只有在它依赖的属性值改变之后，下一次获取 computed 的值时重新调用对应的 get 来计算
computed 适用于计算比较消耗性能的计算场景,将插值表达式或者属性中复杂的计算放到计算属性中编写
有返回值

watch
无返回值，无缓存性，页面重新渲染时值不变化也会执行，watch 更多的是[观察]的作用，类似于某些数据的监听回调，用于观察 props $emit 或者本组件的值，当数据变化时来执行回调进行后续操作

当我们要进行数值计算，而且依赖于其他数据，那么把这个数据设计为 computed
如果你需要在某个数据变化时做一些事情，使用 watch 来观察这个数据变化。

1.22. vue 常用修饰符

.native：可以让组件支持原生事件
.once:　　只触发一次
.prevent　阻止默认事件
.stop　　阻止事件冒泡
.capture　捕获事件，优先执行
.self　　只有点击自己的时候才会触发

.lazy　　当输入框失去焦点的时候，再响应数据的变化
.number　　转换成数字类型

1.23. vue 事件中如何使用 event 对象

```
<button @click="ev($event)">事件对象</button>
```

1.24. 组件传参

父传子
父

```
<template>
    <Zi fname=”jack”></Zi>
</template>
```

子:

```
export default {
props:[“fname”],
mounted(){
    console.log(this.fname);//jack
}
}
```

子传父
父

```
<template>
    {{fname}}
    <Zi @tof.sync=”fname”></Zi>
</template>
export default {
  data(){
   return {
      fname:””
   }
   }
}
```

子

```
export default {
      data(){
        return {
            zname:”jack”
        }
      },
      mounted(){
          this.$emit(“update:tof”,this.zname);//调用自定义事件
      }
}
```

兄弟传参

vue2

```
定义 bus 对象
import Vue from 'vue';
export default new Vue();

使用 bus 传送数据
import bus from '../bus/bus'
bus.$emit("bus01",”jack”);

接收 bus 的数据
import bus from '../bus/bus'
bus.$on(“bus01”,data=>{
      console.log(data);//jack
})
```

vue3

```
安装 mitt 库
npm install mitt


封装 bus
import mitt from ‘mitt’
export default mitt();


使用 bus 传送数据
import bus from '../bus/bus'
bus.emit("bus01",”jack”);

接收 bus 的数据
import bus from '../bus/bus'
```

```
bus.on(“bus01”,data=>{
    console.log(data);//jack
})
```

1.25. v-model 语法糖

v-mode 用在组件上，对应的值会以 value=值得形式传给子组件。并且默认为该组件添加一个@input 事件，用来更新 v-mode 对应变量的值

```
 子组件
<template>
        {{value}}  //页面会渲染出 hello title
</template>
this.$emit(“input”, “helo new title”) //会更新父组件里面的 title 的值

父组件
<text-document v-model=“title”></text-document>
data(){
        return {
                title:”hello title”
        }
}
```

1.26. vue 是函数组件还是类组件，怎么用函数组件

vue 组件默认是类组件，也可以使用函数组件

什么是函数式组件

没有管理任何状态，也没有监听任何传递给它的状态，也没有生命周期方法，它只是一个接受一些 prop 的函数。简单来说是 一个无状态和无实例的组件

函数组件的写法

```
<template functional> ...</template>
```

虽然速度和性能方面是函数式组件的优势、但不等于就可以滥用，所以还需要根据实际情况选择和权衡。比如在一些展示组件。例如，buttons，tags, cards，或者页面是静态文本，就很适合使用函数式组件。

1.27. evenBus 和 vuex 的区别

vuex 可以使用 devTools 监控 state 状态

1.28. vuex 是什么

vuex 是一个状态管理器，里面可以执行异步操作。可以不受组件层级的影响方便的进行数据传输

1.29. vuex 使用流程

1. 在对应模块的 store 中编写 state，mutation 和 action

2. 需要使用数据的组件中引入 mapState,在 computed 中通过...的方式引入使用

3. 需要修改数据的组件中，引入 mapActions,在 methods 中通过...的方式引入方法

然后就可以像调用自己的方法那样子，去修改 store 中的 state 值

1.30. vuex 怎么做持久化

手动利用 HTML5 的本地存储

1、vuex 的 state 在 localStorage 或 sessionStorage 中取值；
2、在 mutations 中,定义的方法里对 vuex 的状态操作的同时对存储也做对应的操作。
这样 state 就会和存储一起存在并且与 vuex 同步

利用 vuex-persistedstate 插件

插件的原理其实也是结合了存储方式,只是统一的配置就不需要手动每次都写存储方法。

```
安装
npm install vuex-persistedstate --save

引入及配置：在 store 下的 index.js 中
import createPersistedState from "vuex-persistedstate"

conststore =newVuex.Store({
            plugins: [createPersistedState({
                        默认存储到 localStorage
                        storage:window.sessionStorage
                        //默认存储所有的 state 数据
reducer(val) {
      return {
        // 只储存 state 中的 assessmentData
        assessmentData: val.assessmentData
      }
   }
          })]
})
```

1.31. pinia 数据持久化

安装插件

```
yarn add pinia-plugin-persistedstate
```

or

```
npm i   pinia-plugin-persistedstate
```

配置插件

```
import { createApp } from "vue";
import App from "./App.vue";
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'
```

```
const pinia = createPinia();
pinia.use(piniaPluginPersistedstate);
createApp(App).use(pinia);
```

开启持久化

```
const useHomeStore = defineStore("home",{
 // 开启数据持久化
 persist: true
 // ...省略
});
```

1.32. 组件的封装与使用

创建一个.vue 文件，通过 props 接收父组件的参数，通过$emit 向父组件传递参数
通过 import from 导入组件，局部使用，或者通过 router 路由配置 path 访问。

1.33. vue 路由传参

第一种：url 传参（参数会显示在地址栏）

```
传参：
this.$router.push("/hello?sname='jack'&saddr=天津");
路由配置:
{path:"/hello",component:Com01}
收参：
let {sname,saddr} = this.$route.query;
```

第二种：params 传参

```
传参：
this.$router.push({name:'hello',params:{sname:'jack',saddr:"田静'});

路由配置：
{name:"hello",component:Com01}

收参：
let {sname,saddr} = this.$route.params;
```

第三种：

```
传参
push({path:'/routerParamPath',query:{sname:'qjack',saddr:'qaddr'}})

路由配置：
{path:"/routerParamPath",component:Com01}

收参：
let {sname,saddr} = this.$route.query;
```

动态路由：
路由到同一个组件，但是每次传递不同的参数，从而让组件展示出不同的效果，叫动态路由

<router-link to="/aa/jack/1"></router-link>
<router-link to="/aa/mary/2"></router-link>
<router-link to="/aa/lucy/3"></router-link>
路由配置
{path:"/aa/:name/:id",component:aa}
组件
let {name,id} = tis.$route.params;

1.34. vue 路由守卫

跟路由相关的一些特殊的钩子函数。分成 3 中类型
全局守卫：所有路由切换，都会触发的守卫
全局前置守卫：路由切换之前调用

```
router.beforeEach((to, from, next) => {
  //第一个参数 to，包含的内容是切换后的路由对象，也就是跳转后的路由对象
  //第二个参数 from，包含的内容的是切换前的路由对象，也就是跳转前的路由对象
  //第三个参数 next()，是否往下执行，如果不写的话路由就不会跳转，操作将会终止
})
```

全局后置守卫：路由切换之后调用

```
router.afterEach((to, from) => {
  //第一个参数 to，包含的内容是切换后的路由对象，也就是跳转后的路由对象
  //第二个参数 from，包含的内容的是切换前的路由对象，也就是跳转前的路由对象
}
```

路由独享守卫：调用指定路由的时候会触发的守卫

```
{
   path: '/routerParams',
   name: 'routerParams',
   component: RouterParam,
   beforeEnter(to,from,next){
     console.log("路由拦截")
     next();
   }
 },
```

组件内的守卫：切换到指定组件路由的时候会触发的守卫

```
export default {
  template: `...`,
  beforeRouteEnter(to, from) {
    // 在渲染该组件的对应路由被验证前调用
    // 不能获取组件实例 `this` ！
    // 因为当守卫执行时，组件实例还没被创建！
  },
```

```
 beforeRouteUpdate(to, from) {
  // 在当前路由改变，但是该组件被复用时调用
  // 举例来说，对于一个带有动态参数的路径 `/users/:id`，在 `/users/1` 和 `/users/2` 之间跳转的时候，
  // 由于会渲染同样的 `UserDetails` 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。
  // 因为在这种情况发生的时候，组件已经挂载好了，导航守卫可以访问组件实例 `this`
 },
 beforeRouteLeave(to, from) {
  // 在导航离开渲染该组件的对应路由时调用
  // 与 `beforeRouteUpdate` 一样，它可以访问组件实例 `this`
 },
}
```

1.35. vue 插槽

正常情况，调用一个组件
<Com>内容</Com>
组件中间的内容不会渲染

只有使用插槽，才能正确的嵌套组件

```
<SlotComponentDest>
<template v-slot:s2="data">
    <p>是否有嫦娥--{{data.sname}}</p>
</template>
<template>
    <p>把酒问青天</p>
</template>
</SlotComponentDest>
```

SlotComponentDest 组件的编写

```
<template>
    <div>
        <h1>dest</h1>
        <p>明月几时有</p>
        <slot/>
        <p>不知天上宫阙</p>
        <slot name="s2" sname="吴刚"/>
    </div>
</template>
```

1.36. 路由模式有几种? 有什么区别?

hash 与 history 两种路由模式

hash 模式的工作原理是 hashchange 事件，可以在 window 监听 hash 的变化。

```
window.onhashchange = function(event){
    console.log(event);
 }
```

hisotry 模式，工作原理需要劫持连接的点击事件，通过阻止默认跳转，渲染页面，修改地址栏的显示地址来实现

1.37. $route 与$router 的区别

$route 表示当前路由对象，可以用来获取参数，获取路由变化
$router 表示路由管理对象，可以用来获取所有路由信息，控制路由跳转

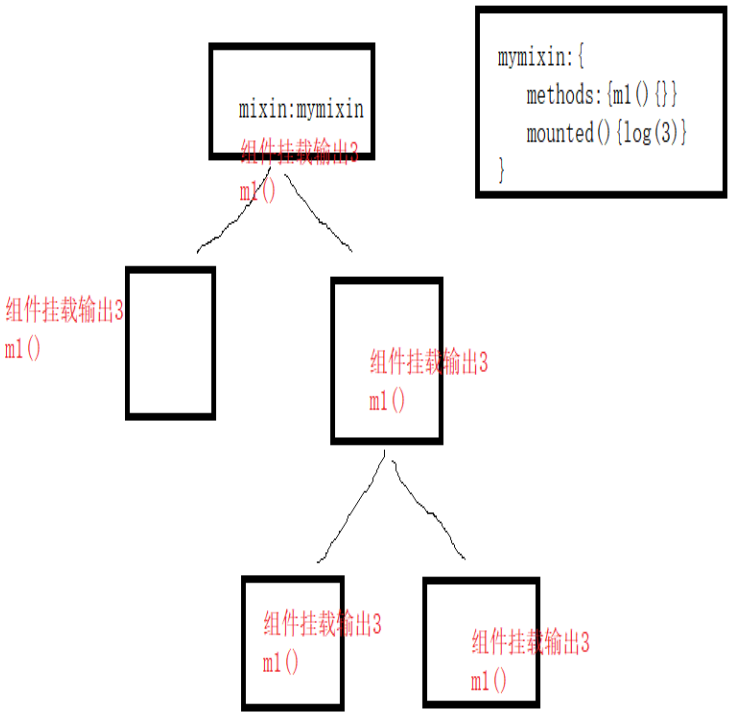
vue3
import {useRoute,useRouter} from 'vue-router'
let $route = useRoute();
let $router = useRouter();

1.38. 虚拟 dom 与真实 dom 的区别

虚拟 dom 的更新不会渲染页面，真实 dom 的更新会导致页面重排重绘。
使用虚拟 dom，最后只更新部分真实 dom，只渲染部分页面。损耗较小。

1.39. mixin 混入

混入是 vue 提供的一项功能。其本质是一个对象。混入的使用，需要先将对象进行注册。分为全局混入和局部混入。混入之后，相当于该混入对象中定义的数据和方法，全部拷贝到了对应组件以及子组件的对应位置上去了。然后在子组件中可以像使用自己的内容那样子去直接使用



1.40. refs 与 ref

$refs 是一个对象，持有已注册过 ref 的所有子组件
ref 被用来给元素或者子组件注册引用信息，引用信息将会注册在父组件的$refs 对象上，如果在普通的 dom 元素上使用，引用指向的就是 dom 元素，
如果用在子组件上，引用就指向组件实例。
$refs 只在组件渲染完成后才填充，并且他是非响应式的，它仅仅是一个直接操作子组件的应急方案--应当避免在模板或者计算属性中使用它

```
<p ref=”myp”> </p>
<button @click=”m1”>点击改变 p 的颜色为红色</button>

m1(){
    this.$refs.myp.style.color=”red”
}
```

1.41. 过滤器与管道

vue 中的过滤器分为两种：局部过滤器和全局过滤器

过滤器主要用在 v-html 指令以及插值表达式中，用来过滤数据，改变数据的显示方式等

当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用！

一个表达式可以使用多个过滤器。过滤器之间需要用管道符“|”隔开。其执行顺序从左往右

1.42. 怎么给 Vue 拓展一个方法

可以通过插件来实现

```
定义插件
        // 创建插件
let chajian = {
        //这个方法的第一个参数是 Vue 构造器，第二个参数是一个可选的选项对象：
        //插件里面所有的内容都要放到 install 里面
        install(Vue, options) {
                // 1. 添加全局方法或 property    hello 添加给了 Vue 这个 class
                Vue.hello = function() {
                        // 逻辑...
                        console.log("插件里面的 hello 方法")
                }

                / 2. 添加全局资源
                Vue.directive('zhi', {
                        bind(el, binding, vnode, oldVnode) {
                                // 逻辑...
                                console.log("全局的指令..zhi")
                        }
                })

                // 3. 注入组件选项
                Vue.mixin({
                        created: function() {
                                // 逻辑...
                        },
                        data(){
                                return {
                                        app:"一剪梅"
                                }
                        }
                })

                // 4. 添加实例方法
                Vue.prototype.$good = function(methodOptions) {
                        // 逻辑...
                        console.log("very good")
```

```
                }
            }
        }

        使用插件
            Vue.use(chajian);

        调用插件内容
        ```
            {{app}}
            <h1 v-zhi>调用指令</h1>
            {{$good() }}

            调用全局方法，必须用 Vue 来调用
            mounted() {
                Vue.hello();   //只能通过 Vue 来调用
            }
        ```
```

1.43. vue-loader 是什么？用途有哪些？

vue-loader 是一个加载器，主要作用解析和转换.vue 文件。提取出其中的逻辑代码 script，样式代码 style,以及 HTML 模板 template，再分别把他们交给对应的 loader 去处理。

1.44. 组件 data 返回一个对象

保持局部作用域
vue 组件中 data 值不能为对象，因为对象是引用类型，组件可能会被多个实例同时引用。如果 data 值为对象，将导致多个实例共享一个对象，其中一个组件改变 data 属性值，其它实例也会受到影响。

data:{}
data(){return {}}

1.45. axios 封装

创建 axios 实例 环境的切换 设置请求超时 post 请求头的设置 请求拦截--设置 token 响应的拦截--获取数据.data 封装 get 与 post

1.46. axios 与 ajax 的区别

axios 是一个使用 ajax 封装的库。在使用上，较 ajax 更方便。而且整合了 promise，能够很好的解决调用的顺序问题。axios 可以设置 baseURL，设置请求拦截，响应拦截，转换请求或者响应数据等。功能比较丰富。在实际项目中，更加受欢迎。

1.47. axios 与 jquery

axios 是一个专门用来发送请求的 js 库,而 jquery 除了发送请求之外,还有很多其他的功能。如果只是为了请求发送，我们肯定首选 axios。而且 axios 结合了 promise，能够很好的解决

请求顺序的问题。

1.48. vue 特效怎么做

原理： 为内容添加 transition 标签，组件在进入，以及离开，都会动态添加不同的 class 的值， 进入的时候有 fade-to,fade-in 等,离开的时候有 leave-active，leave-to ，我们为不同的 class 设置不同的样式，就可以添加对应过渡效果. 具体实现：结合 Animate.ccc 或者 Velocity.js 一起使用

1.49. elementUI

1. 它是什么 2.你为什么要用他 3.你用过他的哪些具体的组件 4.你是在哪些地方用到它的组件的 5.你有没有用过除了 element 之外的其他的 vue 组件(iview,vant ui) varlet

1.50. 动态组件

可以在一个地方，动态展示不同的组件

<component :is="abc"></component>

1.51. 对 mvc 和 mvvm 的理解

MVC: Controller 负责将 Model 的数据用 View 显示出来

MVVM: 要分成 model 层，view 层，viewModel 层

MVVM 的核心是 ViewModel 层，它就像是一个中转站（value converter），
负责转换 Model 中的数据对象来让数据变得更容易管理和使用，
该层向上与视图层进行双向数据绑定，向下与 Model 层通过接口请求进行数据交互，
起呈上启下作用。View 层展现的不是 Model 层的数据，而是 ViewModel 的数据，
由 ViewModel 负责与 Model 层交互，这就完全解耦了 View 层和 Model 层，这个解耦是至关重要的，
它是前后端分离方案实施的最重要一环。

1.52. Vue3.0 新特性

1. 创建实例，使用 createApp，不用 new Vue
2. vue3 主要使用组合式 api，能够按照功能将代码分块
3. vue3 的数据仓库，推荐使用 pinia，而不是 vuex
4. vue3 引入组件，import 之后可以直接使用，不用去注册
5. vue3 写样式的时候，支持引用变量
6. 使用 vite 作为打包工具，速度更快
7. vue3 的体积更小，性能更高
8. vue3 底层采用 ts 编写，跟 ts 兼容的更好

1.53. vue 组件封装

将一些重复性的内容，封装到一个组件里面去，包括 style,template,script 三个部分 ,然后再需要引用组件的地方进行注册，就可以跟标签一样进行调用。

1.54. keep-alive

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染
缓存组件的状态，当组件的内容切换之后，还能退回之前的状态 里面可以设置 include 表示哪些组件要缓存，exclude 表示哪些不适用缓存， max 设置最大缓存的组件实例，会销毁最久没被使用的实例

keep-alive 包含的组件/路由中，会多出两个生命周期的钩子
activated 与 deactivated。

进入组件执行 activated，不会执行 beforeCreate,beforeMounte
离开组件执行 deactivated，不会执行 destroy

1.55. vue 项目首页加载很慢怎么解决

1.路由懒加载
2.组件异步加载
3. 使用异步组件，按需加载
5.图片量多的时候可以进行懒加载
6.CDN 加速
7.在 webpack 打包的过程中，将多余文件去掉，如 map 文件

1.56. vue 和 react 的区别

vue 和 react 同属于 mvvm 框架。都能够为我们的前端项目，提供很好的服务。各自都有丰富的 ui 组件库。唯一不同的，感觉主要就是使用习惯不同。react 更接近原生开发。在 js 代码中，结合 jsx 语法，会频繁的出现 html 标签。但是在 vue 中，html 结构都是在 template 中完成的。需要变化的部分，会结合 vue 的指令或者插值表达式来完成。从使用习惯上来说，个人更喜欢用 vue 来进行开发。

1.57. vue 组件的 scoped 属性的作用

在 style 标签上添加 scoped 属性，以表示它的样式作用于当下的模块，很好的实现了样式私有化的目的；
但是也得慎用：样式不易（可）修改，而很多时候，我们是需要对公共组件的样式做微调的；
解决办法：

①：使用混合型的 css 样式：（混合使用全局跟本地的样式） <style> /* 全局样式 */ </style><style scoped> /* 本地样式 */ </style>
②：深度作用选择器（>>>）如果你希望 scoped 样式中的一个选择器能够作用得“更深”，

例如影响子组件，你可以使用 >>> 操作符：<style scoped> .a >>> .b { /* ... */ } </style>

1.58. vue 是渐进式的框架的理解

我们可以只选用 vuejs 的核心模板引擎来开发，也可以灵活配置路由，仓库，组件系统等来进一步辅助开发。也可以理解成我们在渐进式的使用 vue 的功能。

1.59. vue.js 的两个核心是什么

数据驱动:

双向绑定

组件系统:

.vue 文件

1.60. vue 更新数组时触发视图更新的方法

1. Vue.set

可以设置对象或数组的值，通过 key 或数组索引，可以触发视图更新

obj = Object.assgin({},this.obj,{name:”jack”})

2. Vue.delete

删除对象或数组中元素，通过 key 或数组索引，可以触发视图更新

7.Vue 提供了如下的数组的变异方法，可以触发视图更新

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

1.61. vue 等单页面应用及其优缺点

缺点：

不支持低版本的浏览器，最低只支持到 IE9；

不利于 SEO 的优化（如果要支持 SEO，建议通过服务端来进行渲染组件）；
第一次加载首页耗时相对长一些；

优点：
无刷新,提升了用户体验；
前端开发不再以页面为单位，更多地采用组件化的思想，代码结构和组织方式更加规范化，便于修改和调整,提升了代码的复用性
更好的实现前后端分离，实现 API 共享，用户体验好、内容的改变不需重新加载整个页面。

1.62. 路由懒加载

在路由配置的时候，通过方法的方式来引用组件，从而让项目打包的时候，将组件文件打包到不同的文件中去，最后在需要的时候，才来加载对应组件文件。就是路由懒加载。
如果不采用路由懒加载，项目中所有组件都会一次性打包到一个文件中，页面在加载的时候，体积就会很大，效率较差。
根据实际需要，适当使用路由懒加载，可以提高访问效率。
实现路由懒加载

```
 {
   path:"/transition",
   name:"动画",
   // component:TransitionCom,
   //将路径对应的组件单独放到一个文件中，webpackChunkName 就表示该文件的具体的名字
   component: () => import(/* webpackChunkName: "about" */ '../views/TransitionCom.vue')
 },
```

1.63. 组件异步加载

懒加载-结合 v-if 来实现
需要的时候就加载　　加载该组件对应的文件
不需要的时候就不加载

实现异步加载

```
同步加载
// import Alives from './Alive.vue'

异步加载
const Alives = ()=>import ('./Alive.vue'/*webpackChunkName: 'alive'*/);
```

1.64. 首页加载速度慢

路由懒加载
组件异步加载
cdn 加速

图片懒加载
ssr 服务端渲染

1.65. 图片懒加载

安装

```
npm install vuelazyload --save
```

配置

```
import VueLazyLoad from 'vue-lazyload'
Vue.use(VueLazyLoad);
```

使用

```
<img v-lazy="url" />
```

1.66. ssr 服务端渲染

csr 全称是 client site render,客户端渲染
ssr 全称是 server site render,服务端渲染

渲染：加载代码，显示到页面

正常是客户端渲染（浏览器）,加载 html 结构，然后通过一系列的 ajax 请求，获取数据，最后将数据显示到页面上。渲染完成。

官网：
https://v2.ssr.vuejs.org/zh/guide/universal.html
框架：
https://www.nuxtjs.cn/guide/installation

1.67. 预渲染

可以解决 seo 的问题

1.68. seo 优化

提高网站的搜索排名
办法一：竞价　　裙子　底价 5 块一次
办法二：seo　　通过网站代码优化，提高自然排名
（点击量，）

lighthouce 进行调试

要添加 viewport
编写 title
提供 meta:description
设置合理的关键字
尽可能使用语义化标签
网页结构尽可能简单
图片应该加上 alt 属性说明，对图片做文本解释

单页应用完成 seo
https://zhuanlan.zhihu.com/p/615463087

1.69. vue-cli 提供了几种脚手架模板

vue-cli 的脚手架项目模板有 browserify 和 webpack；主要用 webpack

1.70. 打包工具

gulp 官网：https://www.gulpjs.com.cn/docs/getting-started/creating-tasks/
vite 官网： https://vitejs.cn/

1.71. 如何监听 vue-router 动态路由参数的变化

原来的组件实例会被复用。这也意味着组件的生命周期钩子不会再被调用。你可以简单地 watch（监测变化）$route 对象

1.72. 怎样理解单向数据流

这个概念出现在组件通信。父组件是通过 prop 把数据传递到子组件的，但是这个 prop 只能由父组件修改，子组件不能修改，否则会报错。子组件想修改时，只能通过 $emit 派发一个自定义事件，父组件接收到后，由父组件修改。

1.73. 理解 Vue 中的 Render 渲染函数

Render 函数是 Vue2.x 新增的一个函数、主要用来提升节点的性能，它是基于 JavaScript 计算。使用 Render 函数将 Template 里面的节点解析成虚拟的 Dom 。
Vue 推荐在绝大多数情况下使用模板来创建你的 HTML。然而在一些场景中，你真的需要 JavaScript 的完全编程的能力。这时你可以用渲染函数，它比模板更接近编译器。
简单的说，在 Vue 中我们使用模板 HTML 语法组建页面的，使用 Render 函数我们可以用 Js 语言来构建 DOM。
因为 Vue 是虚拟 DOM，所以在拿到 Template 模板时也要转译成 VNode 的函数，而用 Render 函数构建 DOM，Vue 就免去了转译的过程。

1.74. pinia 的使用流程

创建项目的时候选择 pinia，在项目中，创建一个文件，通过 defineStore 来创建 pinia 仓库，然后导出方法或者变量，在需要使用仓库的组件中引用创建的仓库，直接获取值或者方法调

用即可。

1.75. pinia 与 vuex 的区别

pinia 是 vue3 推荐的状态库
vuex 是 vue2 推荐的状态库
目前 vuex 已经停止了维护。官方推荐使用 pinia。
pinia 跟组合式 api 结合的更加完美
感觉 pinia 比 vuex 的使用更加快捷方便，包括异步操作都很简单。
pinia 比 vuex 更加轻量，快速

1.76. webpack 的打包原理

1.77. vite 的打包原理

1.78. webpack 打包优化

2. Git

2.1. git 是什么

git 是目前世界上最先进的分布式版本控制系统、可以有效、高速地处理从很小到非常大的项目版本管理

2.2. git 的常用命令

```
git init  把这个目录变成 git 可以管理的仓库

git add README.md  文件添加到仓库

git add  不但可以跟单一文件，也可以跟通配符，更可以跟目录。一个点就把当前目录下所有未追踪的文件全部 add 了

git commit -m ‘first commit’把文件提交到仓库

git remote add origin git@github.com:wangjiax9/practice.git //关联远程仓库

git push -u origin master //把本地库的所有内容推送到远程库上

git pull                    //从远程仓库拉取代码
```

2.3.git 与 svn 的区别

Git 是分布式版本控制工具 svn 是集中式版本控制工具

Git 没有一个全局的版本号，而 SVN 有。

Git 和 SVN 的分支不同

git 吧内容按元数据方式存储，而 SVN 是按文件

Git 内容的完整性要优于 SVN

Git 无需联网就可使用（无需下载服务端），而 SVN 必须要联网（须下载服务端）因为 git 的版本区就在自己电脑上，而 svn 在远程服务器上。

2.4.Git 项目如何配置，如何上传至 GitHub。描述详细步骤

1、 注册登录 github

2、 创建 github 仓库

3、 安装 git 客户端

4、 绑定用户信息

5、 设置 ssh key

6、 创建本地项目以及仓库

7、 关联 github 仓库

8、 推送项目到 github 仓库

2.5.git 怎么解决冲突

先更新代码，然后按照提示编辑冲突，最后勾选冲突已解决。然后再次上传。

2.6.如果配置忽略文件

.gitignore