1. 异步处理
2. 常见的方法：（回调函数、 异步监听、 发布订阅者模式）
3. Axios 是什么？怎么使用？ 描述使用它实现登录功能的流程
4. axios 是基于promise 用于浏览器和node的一个http客户端主要用于向后端发送请求
5. 支持promise、提供了一些并发的方法、提供拦截器、提供支持CSRF（跨站请求伪造）
6. Axios fatch ajax 的区别

Axios fatch是支持promise语法， ajax默认使用callback

Fatch 本质上脱离的xhr 是新的语法（默认不传cookie 不能像xhr这样去监听请求的进度）

1. vuex的核心（state、 mutations、getters筛选数据、actions、modules ）
2. Vue-loader 一个加载器，把vue组件转换成js模块
3. 虚拟dom：使用js的方式去添加dom元素，本质上是优化了diff算法
4. MVVM模式和MVC之间的区别：
5. MVC（model模型、 view视图、 controller控制器）

View <——> controller——> model<——> view (<——>可以相互传递；——> 只能单向传递controller——> model)

MVC最大的弱点就是反馈不及时

1. MVVM （model、 view、 viewModel）时刻反馈

View<——>viewModel<——>model

1. keep-alive: 缓存页面（比如从a页面调到B页面，B页面在跳回a页面，a页面不需再去请求数据）
2. include与exclude,允许组件有条件的缓存
3. 实现的原理就是created 的时候将需要缓存的vnode节点，放到cache中，在render中根据name进行获取

理解：

（1）keep-alive可以实现组件的缓存，当组件切换时不会对当前的组件进行刷新，常用的两个属性是include和exclude，常用的两个生命周期是：activated和deactivated。 （LRU算法）

1. vue computed和watch的区别

（1）computed：

a. 支持缓存，只有依赖数据发生改变，才会重新进行计算

b. 不支持异步，当computed内有异步操作时无效，无法监听数据的变化

C. computed 属性值会默认走缓存，计算属性是基于它们的响应式依赖进行缓存的，也就是基于data中声明过或者父组件传递的props中的数据通过计算得到的值

d. 如果一个属性是由其他属性计算而来的，这个属性依赖其他属性，是一个多对一或者一对一，一般用computed

e.如果computed属性属性值是函数，那么默认会走get方法；函数的返回值就是属性的属性值；在computed中的，属性都有一个get和一个set方法，当数据变化时，调用set方法。

(2) watch

a. 不支持缓存，数据变，直接会触发相应的操作；

b.watch支持异步；

c.监听的函数接收两个参数，第一个参数是最新的值；第二个参数是输入之前的值；

d. 当一个属性发生变化时，需要执行对应的操作；一对多；

e. 监听数据必须是data中声明过或者父组件传递过来的props中的数据，当数据变化时，触发其他操作，函数有两个参数，

　　immediate：组件加载立即触发回调函数执行，

deep: 深度监听，为了发现**对象内部值**的变化，复杂类型的数据时使用，例如数组中的对象内容的改变，注意监听数组的变动不需要这么做。注意：deep无法监听到数组的变动和对象的新增，参考vue数组变异,只有以响应式的方式触发才会被监听到。

1. vue响应式数据的原理：
2. object.defineProperty核心
3. 默认vue在初始化数据时，会给data中的属性使用object.defineProperty重新定义所有属性，当页面取到属性时，会进行依赖收集（收集当前组件的watcher），如果属性发生变化会通知相关依赖进行更新操作
4. vue中如何检测数组变化的？
5. 使用函数劫持的方式，重写了数组的方法
6. Vue将data中的数据进行了原型链重写，指向了自己定义的数组原型方法，这样当调用数组api的时候，可以通知依赖更新，会对数组中的引用类型再次进行监听
7. 为何vue采用异步渲染？
8. 因为如果不采用异步更新，那么每次渲染数据都会对当前组件进行重新渲染，所以为了考虑性能，vue会在本轮数据更新后，再异步更新视图
9. nextTick 实现原理？
10. nextTick 主要使用了宏任务和微任务去定义了一个异步方法，多次调用nextTick，会将方法存入队列中，通过这个异步方法清空当前队列，所以nextTick 这个方法就是异步方法
11. watch 中的deep:true是如何实现的？
12. 当用户指定了watch中deep: true时，如果当前监听的值是数组类型，会对对象中的每一项进行求值，此时会将对应的watcher存入到对应属性的依赖中，这样数组发生变化时，就会通知数据进行更新
13. vue的生命周期
14. beforeGreate 在实例初始化之后，数据观测前被调用
15. greate 实例已创建完成调用
16. BerforeMount 在挂载之前开始被调用：相关的render函数首次被调用
17. Mounted
18. beforeUpdate
19. Update
20. beforeDesotry
21. Desotry
22. vue中模板编译原理
23. 将template变化成render函数
24. 为什么v-for和v-if不能连用？
25. v-for的优先级高于v-if，如果连用，会把v-if的每个元素添加一下，会造成性能问题
26. 如何用vnode来描述一个dom节点
27. 虚拟节点就是用一个对象来描述真实的dom
28. vue中的diff算法
29. 先同级比较，在比较子节点
30. 先判断一方有子节点，一方没有的情况
31. 比较都有子节点的情况
32. 递归比较子节点
33. 描述组件的渲染和更新过程？
34. 渲染组件时，会通过vue.extend构建子组件的构造函数，并进行实例化，最终手动调用$mount()进行挂载。更新组件时会进行pathchVnode流程，核心就是diff算法
35. 组件中的data为什么是一个函数？
36. 避免组件之间的数据互相影响
37. 防止被复用
38. 同一个组件被复用多次，会创建多个实例。但这些实例用的是同一个构造函数，如果data是一个对象的话，那么所有组件都共享了同一个对象，为了保证组件数据的独立性，要求每个组件必须通过data函数返回一个对象作为组件的状态
39. vue中的事件绑定原理？
40. vue中的事件绑定分两种：一种是原生的，一种是组件的事件绑定
41. 原生dom绑定采用的是：addEventListener 实现的
42. 组件的事件绑定采用的是：$on 方法
43. v-modal 实现的原理，如何自定义v-modal
44. v-modal可以看成是value+input方法的语法糖
45. 组件的v-modal是value+input方法的语法糖
46. vue中v-html会导致那些问题？
47. 可能会导致xss攻击
48. v-html会替换掉标签内部的子元素
49. vue组件如何通信？单向数据流
50. 父子间通信，父传子通过props、子传父通过$on、$emit(发布订阅)
51. 获取父子组件实例的方法$parent、 $children
52. 在父组件中提供数据，子组件进行消费Provide/inject
53. Ref获取实例的方式调用组件的属性或者方法
54. Event Bus实现跨组件通信（实际就是$on、$emit实现的）
55. Vux状态管理实现通信
56. vue中相同逻辑如何抽离？
57. Vue.mixin用法 给组件每个生命周期，函数等都混入一些公共逻辑
58. 为什么要使用异步组件？

（1）如果组件功能多，打包出的结果会变大，可以采用异步的方式来加载组件。主要依赖import()这个语法，可以实现文件的分割加载



1. 什么是作用域插槽？

插槽：

1. 创建组件虚拟节点时，会将组件的子组件的虚拟节点保存起来，当初始化组件时，通过插槽属性将子组件进行分类

作用域插槽：

1. 作用域插槽在解析时，不会作为组件的孩子节点。会解析成函数，当子组件渲染时，会调用此函数进行渲染（插槽的作用域为子组件）
2. vue父子组件生命周期调用的顺序
3. 加载渲染过程：父（beforeGreate）—>父（Greated）—>父（beforeMount）—>子（beforeGreate）—>子（Greated）—>子（beforeMount）—>子（mounted）—>父（mounted）
4. 子组件更新过程：父（beforeUpdate）—>子（beforeUpdate）—>子(updated)—>父（updated）
5. 父组件更新过程：父（beforeUpdate）—>父（updated）
6. 销毁过程：父（beforeDestroy）—>子（beforeDestroy）—>子（destroyed）—>父（destroyed）

理解：

1. 组件的调用顺序都是先父后子，渲染完成的顺序肯定是先子后父
2. 组件的销毁顺序是先父后子，销毁完成的顺序肯定是先子后父
3. vue中常见的性能优化：
4. 不要将所有的数据都放在data中,data中的数据会增加getter和setter，会收集对应的watcher
5. Vue在v-for时给每项元素绑定事件需要用事件代理
6. 单页面采用keep-alive缓存
7. 拆分组件
8. v-if当值为false时内部指令不会执行，具有阻断功能，尽量使用v-if代替v-show
9. Key保证唯一性（因为模式vue采用的是复用原则）
10. Object.freeza冻结数据（就不会增加getter和setter）
11. 合理使用路由懒加载，异步组件
12. 尽量采用runtime运行时版本
13. 数据持久化的问题（防抖、节流）
14. vue加载性能优化：
15. 第三方模块按需导入（babel-plugin-component）
16. 滚动到可视区域动态加载（vue-virtual-scroll-list）
17. 图片懒加载（vue-Lazyload）
18. 用户体验：
19. app-skeleton 骨架屏
20. app-shell app壳
21. SEO优化：
22. 预渲染插件 prerender-spa-plugin
23. 服务端渲染ssr
24. 打包优化：
25. 使用cdn的方式加载第三方模块
26. 多线程打包 happypack
27. splitChunks 抽离公共文件
28. sourceMap 生成
29. 缓存、压缩
30. 客户端缓存、服务端缓存
31. 服务端gzip压缩
32. vue3.0有哪些改进？
33. vue3采用了ts来编写
34. 支持composition API
35. Vue3响应式原理改成proxy
36. Vdom的对比算法更新，只更新vdom绑定了动态数据的部分
37. 实现hash路由和history路由
38. Onhashchange
39. History.pushState
40. vue-Router中导航守卫有哪些？

完整的导航解析流程：

1. 导航被触发
2. 在失活的组件里调用离开守卫
3. 调用全局的beforeEach守卫
4. 在重用的组件里调用berforeRouteUpdate守卫
5. 在路由配置里调用beforeEnter
6. 解析异步路由组件
7. 在被激活的组件里调用beforeRouteEnter
8. 调用全局的beforeResolve守卫
9. 导航被确认
10. 调用全局的afterEach钩子
11. 触发DOM更新
12. 用创建好的实例调用beforeRouteEnter守卫中传给next的回调函数
13. action和mutation的区别？
14. mutation是同步更新数据（内部会进行是否为异步方式更新数据的检查）（$watch如果是异步变化，在严格模式下会报错）
15. action异步操作，可以获取数据后调用mutation提交最终数据