Vue2相关面试题

一、vue2和react的区别

相同点：

1. 维护核心库，其他功能如路由和全局状态管理交给生态来做；

2. 都有自己的构建工具

3. 都使用了虚拟DOM来提高重绘性能；

4. 都有props的概念，允许组件间的数据传递；

5. 实现组件化应用，提高复用性

不同点：

1. 数据流：Vue默认支持数据双向绑定，而React一直提倡单向数据流

2. vue默认对虚拟DOM具有优化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；React每当应用的状态被改变时，全部子组件都会重新渲染需要手动通过PureComponent/shouldComponentUpdate这个生命周期方法来进行控制

3. vue使用类似常规的HTML模版，react推荐你所有的模板使用JSX书写。

1. Vue2通过Object.defineProperty监听数据变化,vue3通过proxy，具备比较好的性能React监听数据是通过比较引用的方式，需要手动优化

5. react使用hock拓展，Vue2使用mixins，vue3使用hooks

二、vue2和vue3区别

答：生命周期钩子不同、数据双向绑定原理不同、定义变量和方法的方式不同、指令和插槽的使用不同(v-if和v-for不在相互冲突，移除filter、使用v-slot)、vue3支持多个根节点

1. Vue优点 - 轻量级框架大小只有几十kb、易于理解和学习、上手成本低、运行速度更快
2. 什么是MVVM？

答：Model数据模型，View视图层，ViewModel视图数据层，MVVM模式不需要用户手动渲染DOM元素，而是直接将数据绑定到viewModel上面，会自动渲染数据到页面当中，试图变化会通知viewModel层更新数据，viewModel就是我们MVVM模式当中的桥梁，向下负责转换 Model 中的数据对象，向上为View做服务，解耦了 View 层和 Model 层

五、MVVM的优点：降低代码耦合，提⾼视图或者逻辑的重⽤性、利⽤双向绑定数据更新后视图⾃动更新

六、MVVM缺点：调试Bug需要依赖一些工具，ViewModel的构建和维护的成本都会⽐较⾼会影响性能

七：库：将代码集合成一个产品，我们调用库中的方法来实现自己的功能

八、框架是为了解决一类问题而开发的产品，框架是我们在指定的位置编写好代码，框架帮我们调用

九、Vue 单页应用与多页应用的区别

1. SPA单页面应用指只有一个主页面的应用，一开始只需要加载一次js、css等相关资源。所有内容都包含在主页面对每一个功能模块组件化。单页应用跳转切换相关组件刷新局部资源。

2. MPA多页面应用指有多个独立页面的应用，每个页面必须重复加载js、css等相关资源。多页应用跳转，需要整页资源刷新。

十、请说一下vue响应式数据的理解？（Vue的响应式原理）

1. 通过Object.defineProperty/proxy对data的getter/setter方法进行拦截，利用发布订阅者模式，在getter方法中进行订阅，在setter方法中通知，让所有订阅者完成响应。data的每一个属性都新建一个订阅中心作为发布者，对于watcher、computed、template/render作为订阅者，watcher会直接订阅观察监听的属性，对于computed和template/render，如果内部执行获取了data的某一个属性值，就会执行这个属性的getter方法，然后自动完成对该属性的订阅，当属性被修改后，就会执行这个属性的setter方法，从而完成这个属性的发布通知，通知所有的订阅者进行更新。

十一、Vue 是如何实现数据双向绑定的？

数据双向绑定是数据变化更新视图视图变化更新数据，View变化更新Data可以通过事件监听的方式来实现，主要通过以下 4 个步骤来实现数据双向绑定的：

1. 实现一个监听器 Observer：对数据对象进行遍历，包括子属性对象的属性，利用 Object.defineProperty() 对属性都加上 setter 和 getter。这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化。

1. 2. compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图。

3. Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是: ①在自身实例化时往属性订阅器(dep)里面添加自己 ②自身必须有一个update()方法 ③待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调。

4. 订阅器 Dep采用发布-订阅设计模式，用来收集订阅者 Watcher，对监听器 Observer 和 订阅者 Watcher 进行统一管理。

MVVM作为数据绑定的入口整合Observer、Compile和Watcher三者，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

十二、vue中是如何检测数组变化？

vue2当中通过函数劫持的方式重写了数组的方法，利用原型链的特性，每次调用这些方法的时候实际上走的是我们自己重写的方法，vue3当中改用proxy，可以直接监听数组的变化

十三、使用 Object.defineProperty来进行数据劫持有什么缺点？

1. 对象默认只监控自带的属性，新增的属性响应式不生效 (层级过深，性能差)

2. 数组通过索引进行修改 或者 修改数组的长度，响应式不生效

为了解决以上问题，vm.$set -> 修改数组内部使用的是splice方法 和vm.$delete

十四、Proxy与Object.defineProperty优劣对比

1. Proxy 可以直接监听对象而非属性，可以直接监听数组的变化

2. Proxy 有多达13种拦截方法，不限于 apply、ownKeys、deleteProperty

3. Proxy 返回的是一个新对象我们可以只操作新的对象而Object.defineProperty只能遍历对象属性修改

4. Proxy 如果对象内部要全部递归代理，则Proxy可以只在调用时递归，而Object.defineProperty需要在一开始就全部递归，Proxy性能优于Object.defineProperty

5. Object.defineProperty兼容性好支持 IE9，而 Proxy 的存在浏览器兼容性问题而且无法用 polyfill 磨平

十五、怎么理解vue的单向数据流？

父子组件之间的通信是单向的，是为了防止子组件意外的改变父组件的状态使得数据流难以理解，每次父级组件发生更新时，子组件中所有的props都会更新，子组件通过$emit派发一个自定义事件，父组件接收到后，由父组件修改数据

十六、vue数据的丢失 - 初始化时没有定义数据，之后更新的数据是无法触发页面渲染更新的

十七、vue 中给data中的对象属性添加一个新的属性时会发生什么？如何解决？

不会发生试图更新；需要使用$set方法相当于手动的去把新增加的对象属性处理成一个响应式的属性，此时视图也会跟着改变了。

十八、data中某一个属性的值发生改变后，视图会立即同步执行重新渲染吗？

不会立即同步执行重新渲染，Vue实现响应式并不是数据发生变化之后DOM立即变化，而是按一定的策略进行DOM的更新。在更新DOM时是异步的，监测到数据变化Vue 将开启一个队列进行缓冲，在同一事件循环中发生的所有数据变更。如果同一个watcher被多次触发，只会入到队列中一次，因为在缓冲时会去除重复数据，避免不必要的计算和DOM操作，在下一个的事件循环中会刷新队列并执行实际的工作。

十九、在data上面定义了一个$name属性，可以通过Vue.$name拿到吗？

不可以，因为vue内部将所有$开头的属性都默认为是vue本身内置的属性

二十、直接给一个数组项赋值，Vue 能检测到变化吗？

不能，数组考虑性能原因没有用defineProperty对数组的每一项进行拦截，而是选择重写数组方法，在Vue中修改数组的索引和长度是无法监控到的。需要通过上面的方法修改数组才会触发数组对应的watcher进行更新。数组中如果是对象数据类型也会进行递归劫持。可以通过Vue.$set来进行处理, 其核心内部用的是splice方法，针对修改数组的长度使用\*\*vm.items.splice(newLength)\*\*解决

二十一、 Vue中封装的数组方法有哪些，其如何实现页面更新

push,shift,pop,splice,unshift,sort,reverse；首先获取到这个数组的\_\_ob\_\_即Observer对象，如果有新的值就调用observeArray继续对新的值观察变化（也就是通过target.\_\_proto\_\_ == arrayMethods来改变了数组实例的原型）然后手动调用notify通知渲染watcher执行update

二十二、Vue怎么用vm.$set解决数组对象新增属性不能响应 - 通过调用重写的数组splice方法实现

二十三、给响应式数据添加标识为什么不可以直接使用value.\_\_ob\_\_ = this

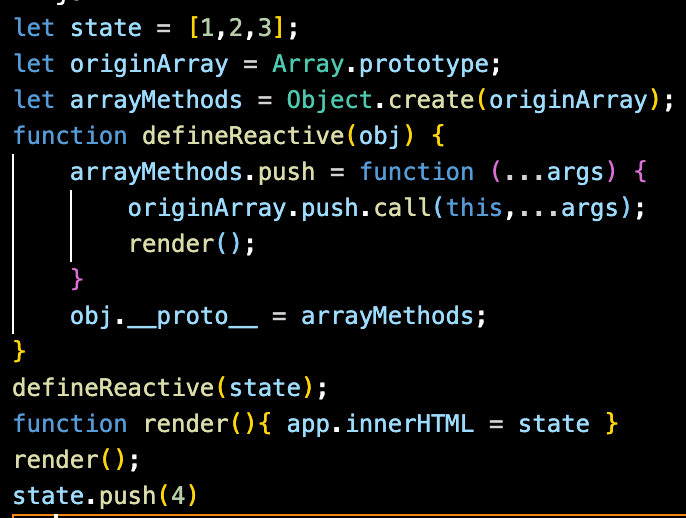
因为value是一个被观测得响应式的值如果这样赋值就会触发Object.defineProperty方法set方法，此时无限递归会陷入死循环，通过Object.defineProperty方法添加\_\_ob\_\_属性这个属性是不能被枚举不能被循环出来的，所以不会有影响

二十四、vue的构造函数为什么没有使用class语法糖

vue构造函数的方法很多，如果使用class进行维护会篇幅很大，对于重写方法的行为也不是很友好，所以还是采用了构造函数的方式，在其原型上增加方法进行拓展

二十五、定义响应式数据的时候在defineReactive的set方法内部，不直接value[key]赋值？会陷入死循环

二十六、手动实现一个proxy的响应式 二十七、Vue是如何重写数组方法的？

二十八、vue为什么采用异步渲染？

vue的数据更新是采用组件级别的更新策略会节省性能，vue会将同一个组件的watcher更新过滤在一起在合适的时机更新，主要操作是：dep.notify通知watcher进行数据更新，然后依次调用watcher的update方法，通过queueWatcher方法将多个属性依赖的相同的watcher整合成一个（每一个watcher都有一个id相同id的整合在一起主要是预防多次更新）最后通过nextTick方法异步清空watcher队列，vue会在数据更新了之后，再去异步更新视图，提高了性能

二十九、Vue中模板编译原理？

模板编译：vue中的模板template无法被浏览器解析并渲染，因为不是正确的HTML语法，所有需要将template转化成一个JavaScript函数，这样浏览器就可以执行这一个函数并渲染出对应的HTML元素，就可以让视图跑起来了。模板编译分为解析parse、优化optimize、生成generate，最终生成可执行函数render。

1. parse阶段：使用大量正则表达式对template字符串进行解析将标签指令属性等转化为抽象语法树AST。
2. optimize阶段：遍历AST找到其中的一些静态节点并进行标记方便在页面重渲染的时候进行diff比较时，直接跳过这一些静态节点，优化runtime的性能。3. generate将最终AST转化为render函数字符串。

三十、template预编译是什么？

预编译在项目构建的编译template的过程。模板编译只会在组件实例化的时候编译一次生成渲染函数之后在也不会进行编译。在项目构建时可以让实际组件在 runtime 时直接跳过模板渲染，进而提升性能

三十一、vue 是怎么解析template的? template会变成什么?

源码会执行 compileToFunctions 将template转化为render函数，调用parse方法将template转化为ast对静态节点做优化，generate将ast抽象语法树编译成 render字符串并将静态部分放到 staticRenderFns 中，最后通过 new Function(``render``) 生成render函数

三十二、模板引擎的实现原理 - 就是\*\*new Function + with\*\*来进行实现的

三十三、computed 什么时候初始化 - vue创建实例的时候

三十四、computed 的缓存是怎么做

computed本身就是一个计算属性类型的watcher，缓存是通过控制watcher实例对象的dirty属性做到的，computed将lazy赋值dirty给一个初始值，让watcher控制缓存的任务开始，如果 computed依赖的数据变化，dirty=true，调用evalute重新计算然后更新缓存值watcher.value；computed的A依赖B，就会将B收集到A的watcher种，B变了就会通知A更新，就会判断dirty = true重新计算否则走缓存

三十五、computed是怎么可以直接使用实例访问到的

在defineComputed函数当中通过`Object.defineProperty(target, key, sharedPropertyDefinition)`将computed的所有属性都定义在了vm上面，所以可以直接使用实例访问到的

三十六、computed是怎样进行计算的

1. 页面更新，读取 computed 的时候，Dep.target 会设置为 页面 watcher。

2. computed被读取，createComputedGetter包装的函数触发，第一次会进行计算

3. computed-watcher.evaluted 被调用，进而computed-watcher.get被调用，Dep.target被设置为computed-watcher，旧值页面 watcher 被缓存起来。computed 计算会读取 data，此时 data 就收集到 computed-watcher，同时 computed-watcher 也会保存到 data 的依赖收集器 dep（用于下一步）。computed 计算完毕，释放Dep.target，并且Dep.target 恢复上一个watcher（页面watcher）

4. 手动 watcher.depend， 让 data 再收集一次 Dep.target，于是 data 又收集到 恢复了的页面watcher，从而使得页面A与数据C关联上

三十七、computed与watch的区别

1. 计算属性computed和监听器watcher都可以观察属性的变化从而做出响应
2. computed本质上就是一个watcher是作为缓存功能的观察者，可以将data的属性进行复杂的计算生成一个新的值提供给渲染函数使用，依赖发生变化computed不会立即重新计算而是先标记为脏数据下次computed被获取会重新计算并返回而watch并不具备缓存性，监听器watch提供一个监听函数，类似于某些数据的监听回调，每当监听的数据变化时都会执行回调进行后续操作

三十八、computed与watch的使用场景

1. 当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed
2. 当我们需要在数据变化时执行异步或开销较大的操作时使用watch，使用watch选项允许我们执行异步操作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，计算属性无法做到的。

三十九、在使用计算属性时，函数名和 data 数据源中的数据可以同名吗？

不可以重名，props，methods 都不可以重名，因为 Vue 会把这些属性挂载在组件实例上，直接使用 this 就可以访问，如果重名就会导致冲突。

四十、watch的原理

watch用于监控用户的data变化，数据变化后会触发对应的watch的回调方法,本质调用的是vue实例的$watch方法，$watch内部创建watcher，如果用户传进来的还有一些特殊的操作如immediate会借助vue响应式原理，默认在取值时会将这个旧值保存下来，然后将watcher存放到对应属性dep中当数据发生变化时通知对应的watcher重新执行getter函数拿到返回的结果作为旧值执行回调函数将新旧值作为参数

四十一、组件中data为什么是一个函数？

因为组件是用来复用但是JS里对象是引用关系如果是对象作用域没有隔离data属性值会相互影响，函数可以保证每个实例可以维护一份被返回对象的独立的拷贝，组件实例之间的data属性值不会互相影响；

四十二、new Vue -data为啥是对象 - new Vue的实例是不会被复用的，因此不存在引用对象的问题。

四十三、 Vue 组件间通信有哪几种方式？

1. 父子组件间通信- A.props/ $eimt B. $refs/$parent C. provide/inject
2. 兄弟组件间通信 - eventBus（$emit / $on)、$parent/$ref
3. 任意组件之间 - eventBus、vuex

四十四、eventBus本质 - 通过创建一个空的 Vue 实例来作为消息传递的对象，通信的组件引入这个实例，通信的组件通过在这个实例上监听和触发事件，来实现消息的传递。

四十五、vue组件会在什么时候下被销毁？

1. 当v-if为false的时候 2. 父组件销毁的时候 3. 调用this.$destory的时候 4. 页面关闭，路由跳转（没有使用keep-alive时的路由切换）

四十六、如何保存页面的当前的状态

2种情况前组件会被卸载和前组件不会被卸载

1. 前组件会被卸载:

A. LocalStorage / SessionStorage

优点：兼容性好，不需要额外库或工具,简单快捷，基本可以满足大部分需求。

缺点：状态通过 JSON 方法储存（相当于深拷贝），如果状态中有特殊情况（比如 Date 对象、Regexp 对象等）的时候会得到字符串而不是原来的值,如果 B 组件后退或者下一页跳转并不是前组件，那么 flag 判断会失效，导致从其他页面进入 A 组件页面时 A 组件会重新读取 Storage，会造成很奇怪的现象

1. 路由传值

优点：简单快捷，不会污染 LocalStorage / SessionStorage;可以传递 Date、RegExp 等特殊对象

缺点：如果 A 组件可以跳转至多个组件，那么在每一个跳转组件内都要写相同的逻辑。

2. 组件不会被卸载

A. 单页面渲染: 要切换的组件作为子组件全屏渲染，父组件中正常储存页面状态。

优点：代码量少,不需要考虑状态传递过程中的错误

缺点：增加 A 组件维护成本,需要传入额外的 prop 到 B 组件,无法利用路由定位页面

B. keep-alive来缓存页面,当组件在keep-alive内被切换时组件的activated、deactivated这两个生命周期钩子函数会被执行 被包裹在keep-alive中的组件的状态将会被保留

四十七、vue.js中如何实现路由嵌套

首先实例化根组件，在根组件中定义组件渲染容器，然后，挂载路由，当切换路由时，将会切换整个页面。

四十八、template和jsx的有什么分别？

都是render的一种表现形式，不同的是：JSX比template具有更高的灵活性复杂的组件中具有优势，template 在代码结构上更符合视图与逻辑分离的习惯，更简单、更直观、更好维护

四十九、组件渲染和更新过程

渲染组件时，会通过 vue.extend() 方法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 patchVnode 流程，核心就是 diff 算法。

五十、为什么要使用异步组件？

节省打包出的结果，异步组件分开打包，采用jsonp的方式进行加载，有效解决文件过大的问题；核心就是包组件定义变成一个函数，依赖import语法，可以实现文件的分割加载。

五十一、谈谈你对 Vue 生命周期的理解？

Vue实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模版、挂载 Dom -> 渲染、更新 -> 渲染、卸载等一系列过程这些函数方便用户在特定阶段有机会添加他们自己的代码

Vue生命周期总共可以分为8个阶段：创建前后, 载入前后, 更新前后, 销毁前后，以及一些特殊场景的生命周期。vue3中新增了三个用于调试和服务端渲染场景

1. beforeCreate：组件实例被创建之初，组件的属性生效之前，在这个阶段只有默认的生命周期函数以及默认的事件，在当前阶段data，methods、computed以及watch上的数据和方法都不能被访问

2. created：组件实例已经完全创建，属性也绑定，但真实dom还没有生成$el还不可用已经完成了数据监测，data和methods初始化完毕可以使用数据更改数据，但是在这里修改的数据不会触发updated函数。可以做一些初始数据的获取，在当前阶段无法与DOM进行交互可以通过vm.$nextTick来访问DOM

3. beforeMount：在挂载开始之前被调用render函数首次被调用，在这之前template模版已经导入渲染函数编译；而当前虚拟dom已经创建完成即将开始渲染，此时修改DOM不会触发updated

4. mounted：在挂载完成后发生el被新创建vm.$el替换，并挂载到实例上去之后调用该钩子此时真实的dom挂载完毕数据完成双向绑定可以访问到真实的dom节点使用$refs属性对dom进行操作

5. beforeUpdate：在更新之前，发生在虚拟DOM打补丁之前，也就是响应式数据更新，虚拟dom重新渲染之前被触发，可以在当前阶段\*\*更改数据\*\*，不会造成重新渲染

6. updated：发生在更新完成之后，当前阶段组件dom已经更新完成

7. beforeDestory：发生在实例销毁之前此时实例完全可以被使用，可以用来清除定时器和复原一些状态

8. destroyed：在实例销毁后剩下了dom空壳。组件已被拆解数据绑定被卸除监听被移出子实例被销毁

keep-alive - activated激活时调用、deactivated组件停用时调用

<keep-alive>包裹动态组件时会缓存不活动的组件实例，而不是销毁它们。<keep-alive>是一个抽象组件，它自身不会渲染一个DOM元素，也不会出现在父组件链中。

五十二、Vue 的父组件和子组件生命周期钩子函数执行顺序？

1. 加载渲染过程：父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted

2. 子组件更新过程: 父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated

3. 父组件更新过程: 父 beforeUpdate -> 父 updated

4. 销毁过程:父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed

五十三、在哪个生命周期内调用异步请求？

可以在钩子函数 created、beforeMount、mounted 中进行调用，因为在这三个钩子函数中，data 已经创建，可以将服务端端返回的数据进行赋值。推荐在 created 钩子函数中调用异步请求可以更快获取到服务端数据，减少页面 loading 时间

五十四、在什么阶段才能访问操作DOM？

在钩子函数 mounted 被调用前，Vue 已经将编译好的模板挂载到页面上，所以在 mounted 中可以访问操作 DOM。

五十五、父组件可以监听到子组件的生命周期吗？

1. 子组件在mounted函数当中通过this.$emit('mounted')触发父组件的事件, 父组件监听到子组件挂载 mounted就绑定处理事件@mounted="something"

2. 在父组件引用子组件时通过 @hook 来监听 \*\*@hook:mounted="doSomething"\*\*, 其它的生命周期事件也可以通过hook的方式

五十六、生命周期钩子是如何实现的?

vue内部主要是使用callHook方法来调用对应的方法。核心是一个发布订阅模式，将钩子订阅好（内部采用数组的方式存储），在对应的阶段进行发布

生命周期两种形式，用户自定义好的生命周期和Vue的全局静态方法Vue.mixin，Vue采用了生命周期函数合并策略使用队列存储好每一类生命周期具备的函数集合，先订阅好后续在特定的时机依次执行即可

五十七、初始化vue实例上面的方法按照props -> methods -> data -> computed -> watch顺序加载

五十八、vue模版当中的优先级？ - render > template > el

五十九、vue 中怎么重置 data并且原理是什么？ - 使用`Object.assgin(this.$data, this.$option.data())`

这个方法就是一个简单的对象合并的方法，`this.$options.data()`就是在组件内部书写的 `data` 函数，执行这个函数就会返回一份初始的 data 数据

六十、你所了解的props是怎样的? props 是子组件访问父组件数据的唯一接口

六十一、 created和mounted的区别

+ created:在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图。

+ mounted:在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作。

六十二、Vue为什么没有类似于React中shouldComponentUpdate的生命周期？

React的Virtual Dom Diff进行差异检测,很多组件肯定不会发生变化需要手动操作来减少diff从而提高程序整体的性能、Vue在一开始就知道那个组件发生了变化并不需要手动控制diff,而组件内部采用的diff方式实际上是可以引入类似于shouldComponentUpdate相关生命周期的,但是通常合理大小的组件不会有过量的diff,手动优化的价值有限

六十三、vue 的keep-alive的作用是什么？（谈谈你对keep-alive的理解）

如果需要在组件切换的时候，保存一些组件的状态防止多次渲染，就可以使用 keep-alive 组件包裹需要保存的组件。keep-alive有以下三个属性：

include 字符串或正则表达式，只有名称匹配的组件会被匹配；

exclude 字符串或正则表达式，任何名称匹配的组件都不会被缓存；

max 数字，最多可以缓存多少组件实例。

keep-alive 包裹动态组件时，会缓存不活动的组件实例。

六十四、keep-alive的流程

1. 判断组件 name 不在 include 或者在 exclude 中直接返回 vnode，说明该组件不被缓存。
2. 获取组件实例 key 如果有获取实例的 key，否则重新生成。
3. key生成规则，cid +"∶∶"+ tag ，不能只有cid因为相同的构造函数可以注册为不同的本地组件。
4. 如果缓存对象内存在，则直接从缓存对象中获取组件实例给 vnode ，不存在则添加到缓存对象中。
5. 最大缓存数量，当缓存组件数量超过 max 值时，清除 keys 数组内第一个组件。

六十五、keep-alive的实现原理

从源码的角度分析组件是自动执行render函数，keep-alive 具体是通过 cache 数组缓存所有组件的 vnode 实例，当include和exclude发生改变时会对cache对象进行遍历取出每一项的name值与新的缓存规则进行匹配，获取组件的key值后去this.cache对象中去寻找是否有该值，如果有命中缓存，如果匹配不上，会将该组件 key 从 keys 数组中删除，然后 push 到 keys数组最后，以便清除最不常用组件。在组件销毁的时候执行pruneCacheEntry函数

附加题：缓存后如何获取数据

beforeRouteEnter、actived

六十六、keep-alive的实现步骤

1. 获取 keep-alive 下第一个子组件的实例对象，通过他去获取这个组件的组件名
2. 通过当前组件名去匹配原来 include 和 exclude，判断当前组件是否需要缓存，不需要缓存，直接返回当前组件的实例vNode
3. 需要缓存，判断他当前是否在缓存数组，存在在原来位置上的 key 给移除，同将组件key放到数组最后面，不存在，将组件 key 放入数组判断当前 key数组是否超过 max 所设置的范围超过那么削减未使用时间最长的一个组件的 key
4. 最后将这个组件的 keepAlive 设置为 true

六十七、keep-alive 本身的创建过程和 patch 过程

缓存渲染会根据 vnode.componentInstance = undefined和 keepAlive 属性判断不会执行组件的created、mounted 等钩子函数，而是对缓存的组件执行 patch 过程直接把缓存的 DOM 对象直接插入到目标元素中，完成了数据更新的情况下的渲染过程。组件的首次渲染判断组件的 abstract 属性，才往父组件里面挂载 DOM，判断当前 keepAlive 和 componentInstance 是否存在来判断是否要执行组件 prepatch 还是执行创建 componentlnstance；prepatch 操作就不会在执行组件的 mounted 和 created 生命周期函数，而是直接将 DOM 插入

六十八、 LRU （least recently used）缓存策略

LRU 缓存策略∶ 从内存中找出最久未使用的数据并置换新的数据。根据数据的历史访问记录来进行淘汰数据，其核心思想是 "如果数据最近被访问过，那么将来被访问的几率也更高"。 最常见的实现是使用一个链表保存缓存数据

1. 新数据插入到链表头部 2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部 3. 链表满的时候，将链表尾部的数据丢弃。

六十九、keep-alive的生命周期

deactivated、activated。同时，beforeDestroy 和 destroyed 就不会再被触发了，因为组件不会被真正销毁。当组件被换掉时，会被缓存到内存中、触发 deactivated 生命周期；当组件被切回来时，再去缓存里找这个组件、触发 activated钩子函数

七十、什么是 mixin

Mixin使Vue 组件编写可插拔和可重用的功能。如果希望在多个组件之间重用一组组件选项，例如生命周期 hook、方法等可以编写为 mixin，并在组件中简单的引用它。然后将 mixin 的内容合并到组件中。如果你要在 mixin 中定义生命周期 hook，那么它在执行时将优化于组件自已的 hook。

附加题：mixin的合并策略

主要是调用merOptions方法，优先递归处理 mixins，先遍历合并parent 中的key，调用mergeField方法进行合并，然后保存在变量options，再遍历 child合并补上parent中没有的key，调用mergeField方法进行合并，保存在变量options，最后通过 mergeField 函数进行了合并

附加题： Vue的几种类型的合并策略

替换型策略有props、methods、inject、computed，就是将新的同名参数替代旧的参数

合并型策略是data, 通过set方法进行合并和重新赋值

队列型策略有生命周期函数和watch，原理是将函数存入一个数组，然后正序遍历依次执行

叠加型有component、directives、filters，通过原型链进行层层的叠加

七十一、vue中mixins的属性/方法/生命周期函数的优先级

1. 对于data定义属性，组件中定义属性覆盖mixins中同名字段

2. 对于created、mounted等生命周期函数，mixins中生命周期函数优先执行（执行顺序按mixins中顺序），再执行组件中生命周期函数

3. 对于methods中的同名方法，组件内的方法覆盖mixins中的方法

注：对于相同的computed属性，组件的computed属性覆盖mixins内的computed属性，而对于相同的watch监听，mixins中的watch监听先执行。

七十二、简述 mixin、extends 的覆盖逻辑

mixin和 extends均是用于合并、拓展组件的，两者均通过mergeOptions方法实现合并

1. mixins 接收一个混入对象的数组，其中混入对象可以像正常的实例对象一样包含实例选项，这些选项会被合并到最终的选项中。Mixin 钩子按照传入顺序依次调用，并在调用组件自身的钩子之前被调用。
2. extends 主要是为了便于扩展单文件组件，接收一个对象或构造函数
3. data/provide：只会将自己有的但是组件上没有的内容混合到组件上，重复定义默认使用组件上的，如果data里面的值是对象，将递归内部对象继续按照这个策略进行合并；
4. props/methods/inject/computed/组件、过滤器、指令属性/el/propsData：只会将自己有的但是组件上没有的内容混合到组件上
5. watch：合并watch监控的回掉方法，执行顺序是将mixins/entends里面的watch定义的回掉函数然后才是组件的回掉函数；
6. Hooks生命周期钩子同一种钩子的回掉函数会被合并成数组，执行顺序是先mixins/entends里面定义的钩子函数，然后才是组件内部的

七十三、mergeOptions 的执行过程

1. 规范化选项（normalizeProps、normalizelnject、normalizeDirectives)

2. 对未合并的选项，进行判断合并处理。

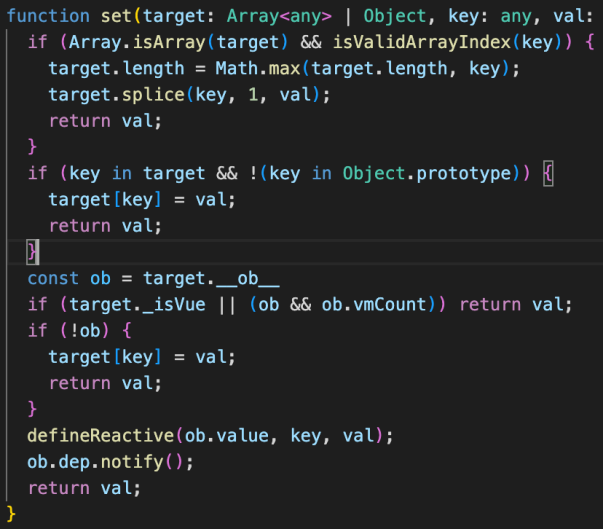
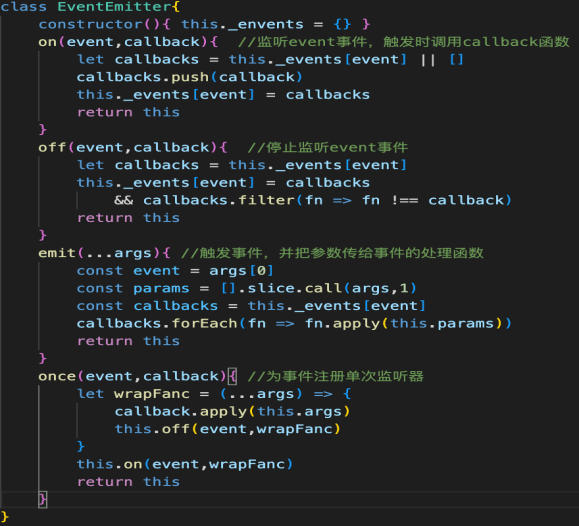
3. 根据一个通用 Vue 实例所包含的选项进行分类逐一判断合并，如 props、data、 methods、watch、computed、生命周期等，将合并结果存储在新定义的 options 对象里。

4. 返回合并结果 options。

七十四、vm.$set 的实现原理是

如果目标是数组，直接使用数组的 splice 方法触发相应式；如果目标是对象，会先判读属性是否存在、对象是否是响应式，最终如果要对属性进行响应式处理，则是通过调用 defineReactive 方法进行响应式处理defineReactive方法就是 Vue 给对象属性采用Object.defineProperty动态添加getter和setter的功能

七十五、手写vm.$set 七十六、手写EventEmitter，实现on/emit/off方法

七十七、$nextTick 原理

nextTick其本质是对 JavaScript 执行原理EventLoop的一种应用。nextTick 的核心是把回调函数放入callbacks等待执行，利用了如Promise 、MutationObserver、setImmediate、setTimeout的原生 JavaScript 方法来模拟对应的微/宏任务的实现，本质是为了利用 JavaScript 这些异步回调任务队列来实现 Vue 框架中自己的异步回调队列，事件循环到了微任务或者宏任务执行函数依次执行callbacks中的回调

nextTick允许开发者在实际项目中使用这个方法来满足实际应用中对DOM更新数据时机的后续逻辑处理

同步更新，则多次对一个或多个属性赋值，会频繁触发 UI/DOM 的渲染，可以减少一些无用渲染

七十八、$nextTick的作用

1. 在数据变化后执行的某个操作，而这个操作需要使用随数据变化而变化的DOM结构的时候，这个操作就需要方法在nextTick()的回调函数中。
2. 如果在created()钩子进行DOM操作也一定要放在nextTick()的回调函数中因为在created钩子函数中页面DOM还未渲染没办法操作DOM想要操作DOM必须将操作代码放在nextTick回调函数中

七十九、怎么给 vue 定义全局的方法？- 将方法挂载到Vue.prototype上面

八十、delete和Vue.delete删除数组的区别

delete 只是被删除的元素变成empty/undefined 其他的元素的键值还是不变。Vue.delete 直接删除了数组改变了数组的键值。

八十一、v-show和v-if有什么区别？

1. v-if是动态的向DOM树内添加或者删除DOM元素；v-show是通过设置DOM的display样式控制显隐

2. v-if切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件；v-show只是简单的基于css切换；

3. v-if是惰性的，如果初始条件为假啥不做；只有在条件第一次变为真时才开始局部编译; v-show是在任何条件下无论首次条件是否为真都被编译然后被缓存而且DOM元素保留；

4.v-if有更高的切换消耗；v-show有更高的初始渲染消耗；v-if适合不怎么改变；v-show适合频繁切换

八十二、v-for和v-if哪个优先级会更高

vue2实践中不应该把v-for和v-if放一起。在vue2中，v-for的优先级是高于v-if，把它们放在一起，输出的渲染函数中可以看出会先执行循环再判断条件，哪怕我们只渲染列表中一小部分元素，也得在每次重渲染的时候遍历整个列表，这会比较浪费；另外需要注意的是在vue3中则完全相反，v-if的优先级高于v-for，所以v-if执行时，它调用的变量还不存在，就会导致异常；通常我们会在过滤列表和避免渲染本应该被隐藏的列表时候联用，此时我们可以通过computed或者包裹一层yemplate来进行避免连用的状况

八十三、v-model是如何实现的，语法糖实际是什么？（v-model的原理是什么？）

1. v-model使用在表单元素上： 动态绑定了input的value指向了message变量，并且在触发input事件的时候去动态把message设置为目标值，（$event.target指代当前触发的事件对象的dom）; text & textarea -> value属性 + input; checkbox/radio -> checked属性和change事件；select -> value作为prop并将change作为事件。

2. v-model作用在组件上：在自定义组件中，子组件通过this.$emit(‘input’,value)来对父组件传值，父组件接受到之后让e.target.value赋值给input中的value从而实现组件内部暴露出组件的值到 v-model所绑定的值中去，本质是原生控件绑定事件捕捉到原生组件的值利用$emit方法触发input方法组件监听到input事件然后把值传入到value中

八十四、v-model 可以被用在自定义组件上吗？如何使用？ - 使用相当于组件绑定value和input事件

八十五、v-if、v-show、v-html 的原理

1. v-if会调用addIfCondition方法，生成vnode的时候会忽略对应节点，render的时候就不会渲染；
2. v-show会生成vnode，render的时候也会渲染成真实节点，只是在render过程中会在节点的属性中修改show属性值，也就是常说的display；
3. v-html会先移除节点下的所有节点，调用html方法，通过addProp添加innerHTML属性，归根结底还是设置innerHTML为v-html的值。

八十六、v-html会导致什么问题？ - XSS攻击、v-html会替换标签内部的元素

八十七、vue.js中常用指令

v-if判断对象是否隐藏；v-for循环渲染；v-bind绑定一个属性（可以简写为：）v-model实现数据双向绑定（支持.trim .number修饰符）、v-once 渲染一次、v-html将字符串转换成dom插入到标签当中、v-show不满足是dom隐藏（不可以使用在template标签上面）、v-on 可以简写成@给元素绑定事件（常用修饰符.stop .prevent .self .once .passive）

八十八、在vue中说说你知道的自定义指令

自定义指令两种：

1. 一种`全局自定义指令`，自定义指令directive方法接收两个参数，一个是`指令名称`，另一个是`函数`；

2. 第二种是`局部自定义指令`，通过组件的`directives`属性定义。

指令定义对象可以提供如下几个钩子函数:

1. bind：只调用一次，指令第一次绑定到元素时调用 bind(el, bindings, vnode, oldVode)，在这里可以进行一次性的初始化设置

2. inserted：被绑定元素插入父节点时调用 inserted(el)（仅保证父节点存在，但不一定已被插入文档中）

3. update：所在组件的VNode更新时调用，但是可能发生在其子VNode更新之前调用。指令的值可能发生了改变，也可能没有。但是可以通过比较更新前后的值来忽略不必要的模板更新

4. componentUpdated：指令所在组件的 VNode及其子VNode全部更新后调用。

5. unbind：只调用一次，指令与元素解绑时调用。

八十九、自定义指令使用场景

1. 普通DOM元素进行底层操作的时候，可以使用自定义指令
2. 自定义指令是用来操作DOM的。尽管Vue推崇数据驱动视图的理念，但并非所有情况都适合数据驱动。自定义指令就是一种有效的补充和扩展，不仅可用于定义任何的DOM操作，并且是可复用的。

九十、自定义指令使用案例

初级应用：鼠标聚焦 、下拉菜单、相对时间转换、滚动动画；高级应用：自定义指令实现图片懒加载、自定义指令集成第三方插件

九十一、vue.js中标签如何绑定事件 - v-on/@语法糖

九十二、简述vue的key的原理

实际开发过程中具备两种途径：1. 当我们在使用v-for时需要给单元加上key 2. 用+new Date生成的时间戳作为key，手动强制触发重新渲染

附加题：v-for当中的key属性的作用是什么？

如果不用key，Vue会采用就地复地原则，设置了key会使 diff 操作可以更准确更快速。Vue 的 diff 过程可以概括为：oldCh 和 newCh 各有两个头尾的变量 oldStartIndex、oldEndIndex 和 newStartIndex、newEndIndex，它们会新节点和旧节点会进行两两对比，即一共有4种比较方式：newStartIndex 和oldStartIndex 、newEndIndex 和 oldEndIndex 、newStartIndex 和 oldEndIndex 、newEndIndex 和 oldStartIndex，如果以上 4 种比较都没匹配，如果设置了key，就会用 key 再进行比较，在比较的过程中，遍历会往中间靠，一旦 StartIdx > EndIdx 表明 oldCh 和 newCh 至少有一个已经遍历完了，就会结束比较。所以Vue中key的作用是：key是为Vue中vnode的唯一标记，通过这个key使我们的diff操作可以更准确、更快速

1. 更准确：因为带 key 就不是就地复用了，在 sameNode 函数 a.key === b.key 对比中可以避免就地复用的情况。所以会更加准确。 2. 更快速：利用 key 的唯一性生成 map 对象来获取对应节点，比遍历方式更快

附加题：用+new Date()生成的时间戳作为key，手动强制触发重新渲染

当拥有新值的rerender作为key时，拥有了新key的Comp出现了，那么旧key Comp会被移除，新key Comp触发渲染

附加题：就地复地原则

最小化element的移动，并且会尝试尽最大程度在同适当的地方对相同类型的element，做patch或者reuse。

九十三、vue事件绑定原理

每一个vue实例都是一个event bus，当子组件被创建父组件将事件传递给子组件，子组件初始化会有一个$on方法将事件注册到内部，在需要时使用$emit触发函数对于原生native事件，使用addEventListener绑定在真实的dom元素上面

九十四、slot是什么？有什么作用？原理是什么？

slot插槽-内容分发机制，组件内部的模板引擎使用slot元素作为承载分发内容的出口。插槽slot是子组件的一个模板标签元素，而这一个标签元素是否显示，以及怎么显示是由父组件决定的。slot又分三类，默认插槽，具名插槽和作用域插槽。

1. 默认插槽：当slot没有指定name属性值的时候一个默认显示插槽，一个组件内有且只有一个匿名插槽。
2. 具名插槽：带有name属性的slot，一个组件可以出现多个具名插槽。
3. 作用域插槽：在子组件渲染作用域插槽时，可以将子组件内部的数据传递给父组件，让父组件根据子组件的传递过来的数据决定如何渲染该插槽。

实现原理：当子组件vm实例化时，获取到父组件传入的slot标签的内容，存放在vm.$slot中，默认插槽为vm.$slot.default，具名插槽为vm.$slot.xxx，xxx 为插槽名name，当组件执行渲染函数时候，遇到slot标签，使用$slot中的内容进行替换，此时可以为插槽传递数据，若存在数据就是作用域插槽。

九十五、过滤器的作用，如何实现一个过滤器filter

filters过滤器是用来过滤数据的改变用户看到的输出，过滤器是一个函数，它会把表达式中的值始终当作函数的第一个参数。过滤器用在插值表达式 {{ }} 和 v-bind 表达式中，然后放在操作符“ | ”后面进行指示

实现原理：

1. 在编译阶段通过parseFilters将过滤器编译成函数调用（串联过滤器则是一个嵌套的函数调用，前一个过滤器执行的结果是后一个过滤器函数的参数）
2. 编译后通过调用resolveFilter函数找到对应过滤器并返回结果
3. 执行结果作为参数传递给toString函数，而toString执行后，其结果会保存在Vnode的text属性中，渲染到视图

九十六、常见的事件修饰符及其作用

.stop：等同于 JavaScript 中的 event.stopPropagation() ，防止事件冒泡；

.prevent ：等同于 JavaScript 中的 event.preventDefault() ，防止执行预设的行为（如果事件可取消，则取消该事件，而不停止事件的进一步传播）；

.capture与事件冒泡方向相反事件捕获由外到内.self会触发自己范围内事件不包含子元素.once触发一次。九十七、使用过 Vue SSR 吗？说说 SSR？

SSR就是在客户端请求服务器的时候，服务器到数据库中获取到相关的数据，并且在服务器内部将Vue组件渲染成HTML，并且将数据HTML一并返回给客户端，而当客户端拿到服务器渲染的HTML和数据之后，由于数据已经有了，客户端不需要再一次请求数据，而只需要将数据同步到组件或者Vuex内部即可。除了数据意外，HTML也结构已经有了，客户端在渲染组件的时候，也只需要将HTML的DOM节点映射到Virtual DOM即可，不需要重新创建DOM节点，这个将数据和HTML同步的过程，又叫做客户端激活。

九十八、ssr的缺点

1. 更多的开发条件限制只支持beforCreate和created两个钩子函数且应用程序需要处于node运行环境
2. 需要更多的服务器负载

九十九、服务端渲染的优点

1. 更好的 SEO: SSR 是直接由服务端返回已经渲染好的页面所以搜索引擎爬取工具可以抓取渲染好页面
2. 首屏加载更快SPA 会等待所有 Vue 编译后的 js 文件都下载完成后，才开始进行页面的渲染，SSR 直接由服务端渲染好页面直接返回显示，无需等待下载 js 文件及再去渲染等，

一百、了解过 style 上加 scoped 属性的原理吗？

scoped属性只属于当下模块局部有效使样式私有化（模块化）不对全局造成污染

scoped 的实现原理：Vue 中的 scoped 属性的效果主要通过PostCSS转译实现

PostCSS 给所有 DOM 添加了一个唯一不重复的动态属性（data-v-\*\*），、然后给 CSS 选择器额外添加一个对应的属性选择器来选择该组件中 DOM，对于第三方组件能通过穿透选择器 ::v-deep 方式来解决

一百零一、vue目录结构assets和static的区别

相同点： assets 和 static 两个都是存放静态资源文件。项目中所需要的资源文件图片，字体图标，样式文件等都可以放在这两个文件下

不相同点：

1. assets 中存放的静态资源文件在项目打包时，也就是运行 npm run build 时会将 assets 中放置的静态资源文件进行打包上传，所谓打包简单点可以理解为压缩体积，代码格式化。而压缩后的静态资源文件最终也都会放置在 static 文件中跟着 index.html 一同上传至服务器。
2. static 中放置的静态资源文件就不会要走打包压缩格式化等流程，而是直接进入打包好的目录，直接上传至服务器。因为避免了压缩直接进行上传，在打包时会提高一定的效率，但是 static 中的资源文件由于没有进行压缩等操作，所以文件的体积也就相对于 assets 中打包后的文件提交较大点。在服务器中就会占据更大的空间。

建议： 将项目中 template需要的样式文件js文件等都可以放置在 assets 中，走打包这一流程。减少体积。而项目中引入的第三方的资源文件如iconfont.css 等文件可以放置在 static 中，因为这些引入的第三方文件已经经过处理，不再需要处理，直接上传。

一百零二、vue实例挂载中发生了什么？

new Vue会调用\_init方法，定义 $set、$get 、$delete、$watch等方法，定义 $on、$off、$emit、$off等事件，定义 \_update、$forceUpdate、$destroy生命周期，调用$mount进行页面的挂载，挂载的时候主要是通过mountComponent方法，定义updateComponent更新函数，执行render生成虚拟DOM，\_update将虚拟DOM生成真实DOM结构，并且渲染到页面中

一百零二、组件和插件的区别？

组件 是用来构成业务模块目标是 App.vue；插件是用来增强你的技术栈的功能模块目标是 Vue 本身

一百零三、什么是虚拟DOM？如何实现一个虚拟DOM？说说你的思路

虚拟DOM以JavaScript 对象 VNode 节点)作为基础的树，用对象的属性来描述节点，最终可以通过一系列操作使这棵树映射到真实环境上

一百零四、Vue3 Diff算法和 Vue2 的区别

+ 编译阶段将事件缓存可以理解为变成静态的了，第一次创建静态节点时保存，后续直接复用，添加静态标记：给节点添加静态标记，以优化 Diff 过程，由于编译阶段的优化能更快的生成虚拟 DOM 以外，还使得 Diff 时可以跳过"永远不会变化的节点"，Diff 优化：Vue2 是全量 Diff，Vue3 是静态标记 + 非全量 Diff并且使用最长递增子序列优化对比流程

VUE3相关面试题

一、对于即将到来的 vue3.0 特性你有什么了解的吗？

Vue3的目标是让 Vue 核心变得更小、更快，因此 Vue 3.0 增加以下这些新特性：

1. 监测机制的改变使用proxy替代Object.defineProperty解决Object.defineProperty只能监测属性，不能监测对象、检测属性的添加和删除、检测数组索引和长度的变更的弊端同时支持 Map、Set、WeakMap 和 WeakSet。新的 observer 还提供了以下特性：a. observable公开的API提供了简单轻量级的跨组件状态管理解决方案 b. 默认采用惰性观察 c更精确的变更通知只有依赖于特定属性的 watcher 才会收到通知 d提供不可变的observable即使是嵌套属性 e.更好的调试功能：我们可以使用新的 renderTracked 和 renderTriggered 钩子精确地跟踪组件在什么时候以及为什么重新渲染。

2. 更改了作用域插槽使用方式，2.x 的机制导致作用域插槽变了，父组件会重新渲染，而 3.0 把作用域插槽改成了函数的方式，这样只会影响子组件的重新渲染，提升了渲染的性能。同时对于 render 函数的方面，vue3.0 也会进行一系列更改来方便习惯直接使用 api 来生成 vdom 。

3. 对象式的组件声明方式：vue2.x 中的组件是通过声明的方式传入一系列 option和ts的结合需要通过一些装饰器的方式来做，虽然能实现功能，但是比较麻烦。3.0 修改了组件的声明方式，改成了类式的写法，这样使得和ts的结合变得很容易

4. 支持自定义渲染器从而使得weex可以通过自定义渲染器的方式来扩展而不是直接fork源码来改的方式。

5.支持 Fragment（多个根节点）和 Protal（在 dom 其他部分渲染组建内容）组件，针对一些特殊的场景做了处理。

6. 基于 treeshaking 优化，提供了更多的内置功能。

附加题：Vue3.0性能提升主要是通过哪几方面体现的？

1. diff算法优化相比vue2增加了静态标记其作用是为了会发生变化的地方添加一个flag标记，下次发生变化的时候直接找该地方进行比较

2.Vue3中对不参与更新的元素，会做静态提升，只会被创建一次，在渲染时直接复用。这样就免去了重复的创建节点，大型应用会受益于这个改动，免去了重复的创建操作，优化了运行时候的内存占用

3.设置事件监听缓存 4.相比Vue2，Vue3整体体积变小了，除了移出一些不常用的API，再重要的是Tree shanking 任何一个函数，如ref、reavtived、computed等，仅仅在用到的时候才打包没用到的模块都被摇掉，打包的整体体积变小 5.vue3采用proxy重写了响应式系统，因为proxy可以对整个对象进行监听，所以不需要深度遍历

附加题：说说Vue 3.0中Treeshaking特性？

是一种通过清除多余代码方式来优化项目打包体积，本质是编译阶段利用ES6 Module判断哪些模块已经加载、判断那些模块和变量未被使用或者引用，进而删除对应代码

二、defineProperty和proxy的区别

Vue在实例初始化时遍历data中的所有属性使用Object.defineProperty 把这些属性全部转为getter/setter。并劫持各个属性getter和setter在数据变化时发布消息给订阅者触发相应的监听回调。问题：1. 初始化时深度遍历对象性能差 2.通知更新过程需要维护大量 dep 实例和 watcher 实例，额外占用内存较多3. 无法监听到数组元素的变化，只能通过劫持重写数组方法 4. 动态新增，删除对象属性无法拦截，只能用特定 set/delete API 代替 5.不支持 Map、Set 等数据结构。Vue3 使用 Proxy 来监控数据的变化。

1. Proxy 直接代理整个对象而非对象属性，这样只需做一层代理就可以监听同级结构下的所有属性变化，包括新增属性和删除属性。
2. 它的处理方式是在 getter 中去递归响应式，这样的好处是真正访问到的内部属性才会变成响应式，简单的可以说是按需实现响应式，减少性能消耗。 3. Proxy 可以监听数组的变化。

三、composition API 与 options API的区别

vue2 采用的就是optionsAPI，易于学习和使用每个代码有着明确的位置但是相似的逻辑, 不容易复用, 在大项目中尤为明显，虽然可以通过mixins 提取相同的逻辑, 但是也并不是特别好维护

vue3 新增的就是compositionAPI，基于逻辑功能组织代码的, 一个功能api相关放到一起，即使项目大了, 功能多了, 也能快速定位功能相关的 api，提升了代码可读性 和可维护性

四、 Composition API与React Hook很像，区别是什么？

React Hook是根据useState调用的顺序来确定下一次重渲染时的state是来源于哪个useState，所以出现了一些限制。1.不能在循环、条件、嵌套函数中调用Hook 2. 必须确保总是在你的React函数的顶层调用Hook 3.useEffect、useMemo等函数必须手动确定依赖关系,而Composition API是基于Vue的响应式系统实现的，区别是：1. 声明在setup函数内，一次组件实例化只调用一次setup，而React Hook每次重渲染都需要调用Hook，使得React的GC比Vue更有压力，性能也相对于Vue来说也较慢 2. Compositon API的调用不需要顾虑调用顺序，也可以在循环、条件、嵌套函数中使用 3. 响应式系统自动实现了依赖收集，进而组件的部分的性能优化由Vue内部自己完成，而React Hook需要手动传入依赖，而且必须必须保证依赖的顺序，让useEffect、useMemo等函数正确的捕获依赖变量，否则会由于依赖不正确使得组件性能下降。

五、 setup函数

setup()函数为使用vue3的 Composition API 新特性提供了统一的入口,setup 函数会在 beforeCreate 、created 之前执行, vue3也是取消了这两个钩子，统一用setup代替, setup相当于一个生命周期函数，vue中过去的data，methods，watch等全部都用对应的新增api写在setup函数中，setup接收两个参数 props 和 context。它里面不能使用 this，而是通过 context 对象来代替当前执行上下文绑定的对象，context 对象有四个属性：attrs、slots、emit、expose，里面通过 ref 和 reactive 代替以前的 data 语法，return 出去的内容，可以在模板直接使用，包括变量和方法

六、setup语法糖 （script setup语法）

<script setup>在单文件组件 (SFC) 中使用组合式 API 的编译时语法糖, 顶层的绑定会自动暴露给模板，所以定义的变量，函数和import导入的内容都可以直接在模板中直接使用

七、reactive、shallowReactive 函数

reactive函数接收一个普通对象，返回一个响应式的数据对象, 相当于Vue2中的Vue.observable API，响应式转换是深层次影响所有嵌套属性，基于proxy来实现

shallowReactive：创建一个响应式代理，它跟踪其自身属性的响应性shallowReactive生成非递归响应数据，只监听第一层数据的变化，但不执行嵌套对象的深层响应式转换 (暴露原始值)

八、 ref和reactive的区别

ref：根据给定的值创建一个响应式数据，返回一个对象，只包含一个value属性，在setup函数内部访问需要加上.value, 用来创建独立的原始值

reactive：将解包所有深层的refs，同时维持ref的响应性。当将ref分配给reactive property 时，ref将被自动解包，reactive/ref的响应式功能是赋值给对象如果给对象解构或者展开, 会让数据丢失响应式的能力

九、ref、 shallowRef 、isRef、toRefs 函数

shallowRef: 浅层ref作用形式，用于大型数据结构的性能优化

isRef: 判断这个值是否由ref创建的toRefs: 可以保证响应式对象当中每一个属性都是响应式的

十、readonly、isReadonly、shallowReadonly函数

readonly: 传入ref/reactive对象，返回一个原始对象的只读代理，对象递归只读

isReadonly: 检查对象是否由readonly创建只读对象 shallowReadonly只处理对象最外层属性响应式只读

十一、 readonly和const有什么区别？

+ const是赋值保护，使用const定义的变量，该变量不能重新赋值。但如果const赋值的是对象，那么对象里面的东西是可以改的。原因是const定义的变量不能改说的是，对象对应的那个地址不能改变

+ 而readonly是属性保护，不能给属性重新赋值

十二、computed、watch函数

computed: 返回一个ref对象，内部传递方法或者一个具有get、set方法的对象

watch：监听特定的数据源，支持多个数据源同时监听、或者监听某一个对象的某一个属性；在回调函数当中执行副作用，懒执行，可以设置immediate

十三、watch 和 watchEffect 的区别

watch作用是对传入的某个或多个值的变化进行监听；触发时会返回新值和老值默认只有变化了才会执行

watchEffect传入一个函数会立即执行；不需要传入监听内容会自动收集函数内部的数据源作为依赖，在依赖发生变化的时候又会重新执行该函数，如果没有依赖就不会执行；不会返回变化前后的新老值

十四、Vue3的生命周期

基本上就是在 Vue2 生命周期钩子函数名基础上加了 on；beforeDestory 和 destoryed 更名为 onBeforeUnmount 和 onUnmounted；然后用setup代替了两个钩子函数 beforeCreate 和 created；新增了两个开发环境用于调试的钩子

十五、setup语法下怎么设置name属性

使用插件vite-plugin-vue-setup-extend；配置vite.config.js的plugins;在script 标签中使用

十六、Vue3怎么让全局组件有提示 - 定义全局的 global.d.ts

十七、 Vue3怎么注册全局自定义指令 - app.directive('指令名‘，配置对象)

vue-router相关的面试题

一、vue-router 路由模式有几种

3种 1. hash模式使用URL hash 值来作为路由 2. history模式依赖 HTML5 History API 和服务器配置

3.abstract 支持所有js运行环境如果发现没有浏览器的 API，路由会自动强制进入这个模式

二、vue-router - hash路由模式实现原理吗？

基于 location.hash来实现的，location.hash值就是URL中#后面的内容，路由模式的实现主要是基于：

1. URL中hash 值是客户端一种状态，当向服务器端发出请求时hash部分不会被发送不能做seo优化
2. hash改变都会在浏览器的访问历史中增加一个记录，能通过浏览器的回退、前进按钮控制hash切换
3. 可以通过 a 标签并设置 href 属性，当用户点击这个标签后，URL 的 hash 值会发生改变；或者使用js来对 loaction.hash 进行赋值改变 URL hash 值
4. 使用 hashchange 事件来监听hash值变化从而对页面进行跳转，window.location.hash.substr(1);

三、history的实现原理

HTML5 提供了 History API 来实现 URL 的变化。通过history.pushState和 history.repalceState这两个 API 可以在不进行刷新的情况下，操作浏览器的历史纪录。pushState是新增一个历史记录repalceState是直接替换当前的历史记录，特性：1. 我们可以使用 popstate 事件来监听 url 的变化，从而对页面进行跳转（渲染）；2. pushState或 replaceState不会触发 popstate 事件，浏览器不会向后端发送请求，这时我们需要手动触发页面跳转（渲染）。popstate 事件的执行是在点击浏览器的前进后退按钮的时候，才会被触发，这两个方法提供了对历史记录进行修改的功能，当这两个方法执行修改时，只能改变当前地址栏的 url，但浏览器不会向后端发送请求

三、在vue项目中如何获取页面的hash变化

1. window.onhashchange监听hash事件window.addEventListener('hashChange', () => {})

2. 通过watch:{ $route:{ handler(newVal,oldVal){ }, deep:true } }

四、hash和history的区别

主要在浏览器的兼容性hash比history更出色，支持低版本浏览器和IE；通过history.pushState存在优势

1. pushState设置的新URL可以是与当前URL同源的任意URL；而 hash 只可修改#后面的部分，因此只能设置与当前 URL 同文档的 URL pushState设置的新 URL 可以与当前 URL 一模一样，这样也会把记录添加到栈中；而 hash 设置的新值必须与原来不一样才会触发动作将记录添加到栈中
2. pushState通过stateObject参数可以添加任意类型的数据到记录中而hash只可添加短字符串

四、什么是前端路由

前端路由是在保证只有一个HTML页面的情况下通过对每个视图展示形式匹配一个特殊的url来实现所谓的切换效果。不会重新向服务端发送请求也不会跳转页面无论是刷新前进还是后退都可以通过特殊url实现。

1. vue-router 路由钩子函数是什么（有哪几种导航守卫）、

钩子函数包括：全局守卫、路由守卫、组件守卫

1. 全局守卫/钩子函数：beforeEach、beforeResolve、afterEach
2. 路由独享守卫：beforeEnter写在路由配置里面只有访问路径才能触发
3. 组件级路由钩子：beforeRouteEnter、 beforeRouteUpdate、 beforeRouteLeave写在组件内部

六、vue-router 路由钩子函数执行完整导航流程

1. 导航被触发 2. 在失活的组件里面调用 beforeRouteLeave 守卫3. 调用全局的 beforeEach 守卫

4. 在重用的组件内部调用 beforeRouteUpdate 守卫（2.2+）5. 在路由配置中调用 beforeEnter

6. 解析异步组件路由7. 在被激活的组件内部调用 beforeRouteEnter

8. 调用全局的 beforeResolve守卫（2.5+）9. 导航被确认10. 调用全局的afterEach钩子

11. 触发DOM更新12. 调用 beforeRouteEter 守卫中传给next的回调函数，创建好的组件实例会作为回调函数的参数传入

七、vue-router 动态路由是什么？有什么问题？

在vue-router的路由路径中支持使用动态路径参数的形式来使用户可以根据不同的业务场景拿到不同的组件入参，支持两种方式：params和query；两种方式的区别

1. query方式 name、path都可以使用params方式只能用name
2. query更加类似ajax当中的get穿参数、params更类似于post，query在浏览器地址栏当中显示参数，params不显示
3. query刷新不会丢失query里面的数据、params刷新会丢失params的数据

八、$route和$router的区别?

$router为VueRouter实例，想要导航到不同URL，则使用$router.push方法；$route为当前router跳转对象，里面可以获取name、path、query、params等

1. vue路由懒加载如何实现？

路由懒加载是将各个模块分开打包，在用户查看下相关模块内容时就直接引入相关模块的打包文件然后进行显示有效的解决了浏览器可能出现短暂时间空白页的情况。懒加载可以分割代码块，提高页面的初始加载效率解决白屏问题。方法1.使用es6的import 2. webpack提供的require.ensure(相同chunkName会被打包成一个js) 3. 使用异步组件resolve

import 是解构过程并且是编译时执行，require 是赋值过程并且是运行时才执行，也就是异步加载，require的性能相对于import稍低，因为require是在运行时才引入模块并且还赋值给某个变量

十、$router.push相当于点击路由链接(可以返回到当前路由界面）队列的方式（先进先出）

十一、$router.replace用新路由替换当前路由(不可以返回到当前路由界面) --> 栈的方式（先进后出）

Vuex相关面试题

一、Vuex为什么要分模块并且加命名空间？

因为vuex使用单一状态树，将所有的状态都集中在一个比较大的对象上面所以将store分割成模块，每个模块拥有自己的state、mutation、action、getter、甚至是嵌套子模块，所以就会出现模块和命名空间，使模块具有更高的封装度和复用性

附加题：vuex划分模块的好处

管理更加容易、保证了store完整的状态树的同时又避免了命名冲突问题，操作state变得更加扁平和直观

二、Vuex和单纯的全局对象有什么区别？

1. Vuex 的状态存储是响应式的，依托于Vue的发布订阅模式，vuex的所有的状态state都通过Vue.set方法处理成了响应式数据

2. 改变store中的状态的唯一途径就是显式地提交mutation

三、为什么mutation中不能做异步操作？

不利于代码调试，每个mutation执行完成后都会对应到一个新的状态变更，这样devtools就可以打个快照存下来记录状态，然后就可以实现 time-travel 了。如果mutation支持异步操作，就没有办法知道状态是何时更新的

四、action和mutation的区别

1. mutations可以直接修改state只能包含同步操作只能通过提交commit调用

2. action是用来触发mutations的，它无法直接改变state可以包含异步操作只能通过store.dispatch触发

五、Vuex有哪几种属性？

有五种，分别是 State、 Getter（state的计算属性）、Mutations 、Actions、 Modules

六、vuex如何异步的修改数据？

在action中通过dispatch分发异步请求，在异步回调的过程中使用commit提交mutation，在mutation当中修改state，使得getters对state的值进行计算封装

七、在模块中，getter和mutation和action中怎么访问全局的state和getter？

1. 在getter中可以通过第三个参数rootState访问，可以通过第四个参数rootGetters访问

2. 在mutation中不可以访问全局的satat和getter，只能访问到局部的state

3. 在action中第一个参数context中rootState访问全局state，rootGetters访问到全局getter

八、vuex的store特性是什么？

vuex内部数据是响应式的，getter被处理成computed具备缓存性，同步修改和异步修改使数据具有跟踪性，模块命名空间的添加在保证store是一个完整的树结构的同时，使得项目的store维护更加便捷，命名也不会存在冲突的问题

九、getters具有什么特性

将state转换成computed进行计算操作可以在多个组件复用具备缓存性多次取值值不变的不会重新取值

十、mutation的特性是什么？

提供commit方法执行用户定义的mutations进行改变状态，commit是一个箭头函数是为了保证当前的this 当前的实例store

十一、谈谈你对vuex的理解

一种状态管理模式，应用遇到多个组件共享状态时可以使用vuex进行管理

附加题：vuex弊端

1. 项目的可维护性会降低，到处修改维护2. 可读性会下降，因为一个组件内部的数据看不出来来自哪里
2. 增加了代码间的耦合度和组件化的初衷是相悖的4. 页面刷新时会使state的数据初始化

附加题；vuex优点 - 响应式的数据管理、结合vue是单向数据流，可以减少组件间的传参地狱

十二、vue当中ajax请求代码应该是写在组件的method中还是vuex的actions中？

如果是被everwhere复用，就可以将请求放在action内部，方便复用，否则放在组件内部

十三、vuex内部数据的处理流程？

页面通过mapAction异步提交事件到action，action通过commit把对应的参数同步提交到mutation，mutation会修改state对应的值，最后通过getter把对应的值跑出去，在页面的计算属性中通过mapGetter来动态获取state当中的值

十四、vuex插件的机制是什么？

就是一个函数接收store作为唯一参数在Vuex.Store构造器选项plugins引在store/plugin.js文件中写入

附加题：在Vuex插件中怎么监听组件中提交mutation和action？

1. 用Vuex.Store的实例方法subscribe监听组件中提交mutation

2. 用Vuex.Store的实例方法subscribeAction监听组件中提交action 在store/plugin.js文件中写入

十五、vuex使用actions时不支持多参数传递怎么办？ - 放在对象里面进行传递

十六、vuex怎么知道state是通过mutation修改还是外部直接修改的

通过$watch监听mutation的commit函数中\_committing是否为true；严格模式下不允许直接修改

十七、怎么监听vuex数据的变化？ - 先用计算属性获取state、然后再通过watch进行监听

十八、你有使用过vuex的module吗？主要是在什么场景下使用？

把状态全部集中在状态树上，非常难以维护。按模块分成多个module，状态树延伸多个分支，模块的状态内聚，主枝干放全局共享状态，主要是在跨团队间维护公共项目中使用

十九、页面刷新后vuex的state数据丢失怎么解决？vuex-persistedstate插件createPersistedState()方法

二十、vuex中状态是对象时，使用时要注意什么？

因为对象是引用类型，复制后改变属性还是会影响原始数据，这样会改变state里面的状态，是不允许，所以先用深度克隆复制对象，再修改。

二十一、怎么在组件中批量使用Vuex的state状态？

使用mapState辅助函数, 利用对象展开运算符将state混入computed对象中

二十二、Vuex中要从state派生一些状态出来，且多个组件使用它，该怎么做？

使用getter属性，相当Vue中的计算属性computed，只有原状态改变派生状态才会改变。

二十三、怎么在组件中批量给Vuex的getter属性取别名并使用 - mapGetters 传递对象进行设置别名

二十四、Vuex中action通常是异步的啥结束呢 - action函数中返回Promise再提交时候用then处理

二十五、 Vuex中有两个action，分别是actionA和actionB，其内都是异步操作在actionB要提交actionA需在actionA处理结束再处理其它操作，怎么实现 - 利用ES6的async和await来实现。

二十六、在模块中，getter和mutation接收的第一个参数state，是全局的还是模块的？ - 局部的state。

二十七、在组件中怎么访问Vuex模块中的getter和state - this.$store.getters / this.$store.state

二十八、怎么提交mutation和action？this.$store.commit / this.$store.dispatch

二十九、怎么在带命名空间的模块内提交全局的mutation和action - 将 { root: true } 作为第三参数传给 dispatch 或 commit 即可

三十怎么在带命名空间的模块内注册全局的action - 增加root: true

三十一、提交modules命名空间moduleA中mutationA - this.$store.commit('moduleA/mutationA',data)

三十二、Vuex的严格模式是什么,有什么作用,怎么开启

在严格模式下，无论何时发生了状态变更且不是由 mutation函数引起的，将会抛出错误。这能保证所有的状态变更都能被调试工具跟踪到

三十三、v-model上怎么用Vuex中state的值？ - 需要通过computed计算属性来转换

三十四、在组件中多次提交同一个action/mutation - 使用mapActions/mapMutations辅助函数

Pinia面试题

一、什么是pinia？

pinia代替Vuex的一款轻量级状态管理库，设计理念是让Vue Store拥有一款Composition API方式的状态管理库，并同时能支持 Vue2.x版本的Option API 和 Vue3版本的setup Composition API开发模式，并完整兼容ts写法适用于所有的vue项目。

二、pinina相对于vuex的优点是什么？

1. 体积小大约为1kb，2. 完整的ts支持 3. 没有模块嵌套命名空间，可以创建多个store来进行统一的管理数据 4. 去除 mutations 5. 支持 Vue devtools

三、pinia与vuex代码分割机制的差别

vuex代码分割：在打包时，如果存在多个store的情况下，会将多个store合并打包，当页面引用vuex的时候，这个包会引入到这个页面一起打包，最后输出一个js的chunk块，这样的问题在于如果当前这个页面只需要其中一个store，但是其他无关的store也会被打包进来，造成资源的浪费

pinia的打包分割： 在打包的过程中，pinina会检查引用依赖，当页面使用到其中一个store，在打包的过程中只会把用到的store和页面合并输出1个js chunk，其他store不会耦合在其中，pinina能做到这一点，是因为它本身的设计就是store分离的，解决了项目的耦合问题

四、pinia在setup模式下的调用机制是什么？先调用pinia的install方法然后在调用store的具体的信息

五、pinia如何实现在打包过程中与业务代码解耦？

为了让appStore实例与项目解耦，在构建时要把appStore抽取到公共chunk，将pinia的全局库实例打包进vendor，避免和页面一起打包造成资源重复引入

六、pinia在拿到store当中的state如何保持其响应式行为？ - 使用pinia提供的storeToRefs进行转换

七、如何实现pinia的数据持久化？ - 插件 pinia-plugin-persist 可以辅助实现数据持久化功能。

八、createPinia的实现原理

createPinia主要的用途就是用来初始化并收集所有的store以及一些其他信息暴露在全局

1. 首先创建一个pinia的对象，需要使用markRow标记，防止pinia被再次的做响应式处理，不让它变成响应式的，即`const pinia = markRow({})`

2. vue在使用的过程中调用use方法创建store，所以pinia对象提供install方法，install的主要用途在于将pinia实例暴露到app上（通过provide通讯方式），所有组件都可以inject注入使用，保证vue2里也可以通过$pinia使用，将app保留一份在pinia上

3. 为了所有的store统一进行管理，创建一个scope独立空间，run方法的返回值就是回调函数fn的返回值，将state、\_e(scope)和\_s记录所有store放到实例上，还有保存着app的\_a

Webpack相关面试题

一、使用 webpack 时如何优化项目体积

1. 多入口的情况下使用CommonsChunkPlugin来提取公共代码使用SplitChunksPlugin插件提取公共第三⽅库，利用浏览器缓存可以长期缓存这些无需频繁变动的公共代码

2. 使用Tree-shaking剔除多余代码在启动webpack时追加参数--optimize-minimize，使用Code Splitting将代码按路由维度或者组件分块可以按需加载的同时充分利⽤浏览器缓存，使用ParallelUglifyPlugin和UglifyJsPlugin进行js压缩，利⽤cssnano来压缩css，代码还可以使用gzip或者brotli压缩

3.利⽤CDN加速将引⽤的静态资源路径修改为CDN上对应的路径

4. splitChunksPlugin把运行时被引用多次的库进行分包此时分为多个chunk虽不能把总体积变小但可提高加载性能

5. 控制babel/postcss时采用browserlist，越先进的浏览器所需要的 polyfill 越少，体积更小

附加题：如何借助webpack优化前端的性能

1.使用terser进行js代码压缩2. 使用css-minimizer-webpack-plugin压缩css代码3.HtmlWebpackPlugin插件来生成HTML的模板时候通过配置属性minify进行html优化 4. compress-webpack-plugin对文件的大小进行压缩减少http传输过程中宽带的损耗 5. 使用tree-shaking消除js死代码，purgecss-plugin消除css无用的代码 6. 按需加载文件splitChunksPlugin可以分出更小的bundle，以及控制资源加载优先级，提供代码的加载性能 7. 可以通过InlineChunkHtmlPlugin插件将一些chunk的模块内联到html

附加题：如何提高webpack的构建速度

1. 优化loader配置：通过配置include、exclude、test属性来匹配文件 2. 合理使用rsolve.extensions:引入文件的时候若没有文件后缀名则会根据数组内的值依次查找3. 优化rsolve.modules; 指定绝对路径以减少寻找的过程 4.使用resolve.alias减少文件目录的查找过程 5. 使用动态的DLLPlugin插件将不常改变的代码抽成一个共享的库，在之后的编译的过程中会被引入到其他的代码中 6. 使用cache-loader: 将结果缓存在磁盘中显著的提升二次构建时的速度 7. 使用terser/thread-loader启动多线程进行编译 8. 合理的使用sourcemap 9. 使用swc-loader优化AST转换的loader

二、webpack的工作流程了解吗？

Webpack是一个前端资源加载/打包工具。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则变成js，最终实现模块化

1. 初始化流程
2. 初始化参数：解析webpack配置参数，合并shell传入的和webpack.config.js文件配置的参数，形成最后的配置结果，各个配置项拷贝到 options 对象中
3. 初始化Compiler编译对象，为webpack声明周期不执行具体的任务，只是做调度工作
4. 加载所有配置插件plugin，监听webpack构建生命周期的事件节点做出相应的反应
5. 编译构建流程
6. 执行Compiler对象的run方法开始执行编译
7. 根据配置中的entry找出入口文件从入口文件出发开始解析文件即执行before-resolve和build module，构建AST语法树调用所有配置的Loader对模块进行编译（即执行normal-module-loader以及programe）再找出该模块依赖的模块再递归本步骤直到所有入口依赖文件都经过了处理
8. 根据入口和模块之间的依赖关系组装成一个个包含多个模块的Chunk再把每个 Chunk 转换成一个单独的文件加入到输出列表，执行seal方法并开始优化操作，在确定好输出内容后根据配置确定输出的路径和文件名把文件内容写到文件系统（即执行dist）
9. Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用Webpack 提供的 API 改变 Webpack 的运行结果
10. webpack编译构建流程即Complier -> run方法主要流程
11. compile开始编译：构建Compilation对象执行模块创建、依赖收集、分块、打包等等
12. 然后根据入口文件分析模块和依赖的模块来创建这些模块对象
13. 根据入口依赖使用工厂函数创建一下空的module对象
14. 执行buildModule，调用使用的loader完成模块的转换使用acorn解析转换后的内容输出AST，从配置的入口分析AST，当遇到require语句后将其加入到依赖的模块列表同时对新找出的这个模块再次分析

四、webpack 中的loader的作用是什么？

webpack只能处理JS和JSON文件，loader的是用来处理其他类型的文件转换成js文件。loader是一个导出为函数的JavaScript 模块，它接收上一个 loader 产生的结果或者资源文件作为入参。可以使用多个loader串联由loader-runner执行，compiler得到最后一个 loader 产生的处理结果进行编译。

五、loader的执行机制是什么？

六、有没有自己写过loader

没有,看过less-loader源码内部定义一个函数内部执行less.render函数在回调函数当中调用this.async

七、webpack常见的loader

1. file-loader：把⽂件输出到⼀个⽂件夹中，在代码中通过相对URL去引⽤输出的⽂件

2. url-loader：能在⽂件很⼩的情况下以base64的⽅式把⽂件内容注⼊到代码中去

3. source-map-loader加载额外的Source Map⽂件⽅便断点调试4. image-loader加载并且压缩图⽚⽂件

5. babel-loaderES6 转换成 ES5 6. css-loader：加CSS⽀持模块化、压缩、⽂件导⼊等特性

7. style-loader把CSS代码注⼊js中通过DOM操作去加载CSS 8.eslint-loader通过ESLint检查js代码

八、webpack配置多个loader什么顺序？

从右向左，最右侧的loader读的是源文件、最左侧的loader一定是返回一个js模块

九、引入自定义loade的三种方式？

1. 在webpack.config.js当中配置resolveLoader.alias 2. webpack.config.js当中resolveLoader.modules 3. 使用loader的绝对路径，即通过正则匹配文件，通过use使用loader

十、loader的返回值是固定的吗？是返回js的源代码还是返回ast语法树？、

最左边的loader返回的必须是源代码，因为它的返回值是给webpack，webpack要用它来生成ast树

其他的loader的返回值没有要求，可以是任意内容，但是必须是下一个loader能处理的内容

十一、url-loader和file-loader的区别？

url-loader是在file-loader的基础上增加了配置项limit小于limit设置的数值的时候url-loader将文件转换成base64的形式，反之和url-loader的功能一样

十二、css-loader的作用 - 处理css当中的@import语法以及url('./images/logo.png)语法

十三、less-loader的作用 - 是将less的语法转换成css语法

十四、style-loader - 把css->JS脚本就是动态创建style标签并且把这个标签插入到HTML里的header

十五、webpack和babel之间的关系

webpack在编译时遇到js文件会调用babel-loader进行文件内容转换在babel转换时用babel插件

十六、webpack内部使用的babel吗？不是，使用的是acorn

十七、vue-loader 的实现原理是什么

vue-loader会把sfc中的内容拆分为 template，script，style 三个虚拟模块，然后分别匹配 webpack 配置中对应的 rules，template中的内容会通过vue compiler转换render函数后合并到script虚拟模块中。

scoped style 会经过 vue-loader/style-post-loader 的处理，成为只匹配特定元素的私有样式。

附加题：vue-loader的特点

1. 它可以处理我们写的VUE单文件组件2.vue文件能被拆解成js和css两个部分，能被浏览器理解

十八、打包器(webpack/rollup) 如何加载 json 等非 Javascript 资源？加载json：json-loader

十九、打包器(webpack/rollup) 如何加载 style 样式资源？

1. css-loader (opens new window): 处理 CSS 中的 url 与 @import，并将其视为模块引入，此处是通过 postcss 来解析处理2. style-loader (opens new window): 将样式注入到 DOM 中，使用style-loader，创建一个 style 标签，并且把样式资源写到这个 style 的 innerHTML 中，使用 DOM API 加载 CSS 资源，使用mini-css-extract-plugin将CSS单独抽离出来提高性能

二十、什么是polyfill？

降级/替代方案，将ES6+的API转换为在低版本浏览器上可以实现相同功能的替代实现。比如如果浏览器不支持Promise，那我们可以用setTimeout来替代实现。

二十一、如何设置polyfill？

1.使用@babel/preset-env但是要对useBuiltIns进行配置2. babel插件@babel/plugin-transform-runtime

二十二、polyfill的最佳实践

一种是项目当中的配置的写法一种是类库的配置写法都是使用@babel/preset-env的useBuiltIns搭配@babel/plugin-transform-runtime来搭配实现的

二十三、webpack 实现插件机制

Webpack本质上是一种事件流的机制，webpack在其内部对象上创建各种钩子函数，插件将自己的方法注册到对应钩子上交给webpack，webpack工作流程就是将各个插件串联起来在编译过程中，插件通过事件和注册和监听触发webpack生命周期中的函数方法

二十四、 webpack 常见的plugin有哪些

1. ProvidePlugin自动加载模块，代替require和import 2. html-webpack-plugin根据模板自动生成html代码并自动引用css和js文件 3. extract-text-webpack-plugin 将js文件中引用样式单独抽离成css文件

4. DefinePlugin编译时配置全局变量5. HotModuleReplacementPlugin热更新6. happypack通过多进程模型，来加速代码构建7. optimize-css-assets-webpack-plugin不同组件中重复的css可以快速去重

8. webpack-bundle-analyzer文件分析工具将bundle文件以可交互缩放的treemap的形式展示。

9. compression-webpack-plugin`生产环境可采用gzip压缩JS和CSS 10. clean-wenpack-plugin清理每次打包下没有使用的文件

二十五、webpack 插件如何实现

1. webpack本质是一个事件流机制，核心模块：tabable(Sync + Async)Hooks 构造出 === Compiler(编译) + Compiletion(创建bundles)2. 在启动webpack时一次性建立compiler对象即完整的webpack环境配置当应用一插件时插件将收到compiler对象的引用。3. 创建生成一个新的编译资源。compilation对象表现了当前的模块资源、编译生成资源、变化的文件、以及被跟踪依赖的状态的信息。compilation对象也提供了很多关键时机的回调，以供插件做自定义处理时选择使用4. 创建一个插件函数，在prototype上定义apply方法指定一个webpack自身事件钩子5. 函数内部处理webpack内部实例特定数据 6. 处理完成后调用webpack提供的回调函数

二十六、webpack 离线缓存静态资源如何实现

在配置webpack时使用html-webpack-plugin来注入到和html一段脚本来实现将第三方或者共用资源进行静态化存储在html中注入一段标HtmlWebpackPlugin.options.loading.html ,loader中即可通过配置html属性将script注入进去，通过配置 webpack-manifest-plugin 生成 manifestjson文件用来对比js资源的差异，做到是否替换，也要写缓存script；在我们做Cl以及CD的时候，也可以通过编辑文件流来实现静态化脚本的注入，来降低服务器的压力，提高性能。

二十七、插件有执行顺序吗

不同的hook，触发的顺序就是hooks触发的顺序，webpack具有自己规定好的钩子执行顺序，同一个hook，就是注册的顺序

二十八、loader和plugins的区别

loader加载器将浏览器不识别的文件处理成js文件，loader作为模块的解析规则对于不同文件使用什么相应的loader进行解析；plugin插件机制拓展webpack的功能，使webpack更灵活性，plugin会监听webpack的运行生命周期期间会广播出很多事件，在合适的时机通过webpack的API改变输出结果

二十九、插件什么时候挂载 - 插件的挂载或者说监听是在编译启动前全部挂载的

三十、刚刚提到了plugin，现在有两个plugin，plugin1可以派发事件让plugin2监听吗？

Webpack 启动后在会先创建plugin的实例。 在初始compiler对象后再调用Plugin.apply(compiler) 给插件实例传入 compiler 对象。 插件实例在获取到compiler对象后就可以通过 compiler.plugin监听到广播出来的事件并且可以通过compiler对象去操作 Webpack，此时是可以广播出新的事件，所以在新开发的插件中也能广播出事件，给其它插件监听使用。

三十二、webpack 的热更新原理

HMR是指无需刷新在内存环境中即可替换掉过旧模块，热模块替换的原理时通过chunk的方式加载最新的modules找到\_\_webpack\_\_modules\_\_中对应的模块逐一替换，并删除其上下缓存

三十三、webpack 的热更新流程

1. webpack 对文件系统进行 watch 打包到内存中开启了express应用，添加了对webpack编译监听

2. devServer 通知浏览器端文件发生改变，在启动devServer时sockjs 在服务端和浏览器端建立了一个 webSocket长连接,当对应模块发生变化的时候会生成两个文件.manifast和.js,然后webpack编译和打包各个阶段状态告知浏览器不用刷新网页而是刷新某个模块3. webpack-dev-server/client接收服务端消息做出响应4. webpack 接收到最新hash值验证并请求模块代码5. HotModuleReplacement.runtime对模块进行热更新6. 调用accept方法及时将新的内容插入页面

附加题：服务端的工作原理

1. 启动 webpack-dev-server服务器2. 创建webpack实例3. 创建server服务器4. 添加webpack的done事件回调5. 编译完成向客户端发送消息6. 创建express应用app7. 设置文件系统为内存文件8. 添加webpack-dev-middleware中间件9. 中间件负责返回生成的文件10. 启动webpack编译11. 创建http服务器并启动服务12. 使用sockjs在浏览器端和服务端之间建立一个websocket长连接13. 创建socket服务器

附加题：客户端的工作原理

1. webpack-dev-server/client端会监听到此hash消息2. 客户端收到ok消息后会执行reloadApp方法进行更新3. 在reloadApp中会进行判断，是否支持热更新，如果支持的话发生 webpackHotUpdate事件，如果不支持就直接刷新浏览器4. 在 webpack/hot/dev-server.js 会监听 webpackHotUpdate 事件5. 在check方法里会调用module.hot.check方法6. HotModuleReplacement.runtime请求Manifest7. 通过调用 JsonpMainTemplate.runtime 的 hotDownloadManifest方法8. 调用JsonpMainTemplate.runtime的hotDownloadUpdateChunk方法通过JSONP请求获取最新的模块代码9. 补丁js取回来或会调用 JsonpMainTemplate.runtime.js 的 webpackHotUpdate 方法10. 然后会调用hotAddUpdateChunk方法动态更新模块代码11. 然后调用hotApply方法进行热更新

三十四、webpack如何实现持久化缓存

1. 服务端设置http缓存头（cache-control）2. 打包依赖和运行时到不同的chunk，即作为splitChunk, 因为他们几乎是不变的3. 延迟加载：使用import()方式，可以动态加载的文件分到独立的chunk,以得到自己的chunkhash4. 保持hash值的稳定：编译过程和文件内通的更改尽量不影响其他文件hash的计算，对于低版本webpack生成的增量数字id不稳定问题，可用hashedModuleIdsPlugin基于文件路径生成解决

三十五、webpack treeShaking机制的原理

摇树优化基于ES Module进行静态分析通过移除多余代码来优化打包体积的，生产环境默认开启。可以在代码不运行的状态下分析出不需要的代码；利用es6模块的规范将ES6 Module引入进行静态分析，故而编译的时候正确判断到底加载了那些模块并且静态分析程序流，判断那些模块和变量未被使用或者引用，进而删除对应代码

三十六、对于已经 import 但未实际使用的模块使用 webpack 还会对它打包吗？

会对它进行打包，webpack 打包时会将整个类进行标记，tree shaking只处理顶层内容，例如类和对象内部都不会再被分别处理，如果模块没有导入但是模块内的函数存在副作用也会被打包,解决方法是在模块所在的 npm 包的 package.json 中增加 sideEffects: false, 表示所有的模块/文件都是没有副作用的，或者有副作用的话被删了也没关系

三十七、使用 webpack 如何分包 - 4.0之前是CommonsChunkPlugin 4.0 optimization.splitChunks

三十八、code Splitting 的原理是什么？

附加题：webpack 中的 code spliting 是如何动态加载 chunk 的

三十九、webpack 的 runtime 做了什么事情？

最后生成的代码，1. \_\_webpack\_modules\_\_: 维护一个所有模块的数组。将入口模块解析为 AST，根据 AST 深度优先搜索所有的模块，并构建出这个模块数组。每个模块都由一个包裹函数 对模块进行包裹构成。2. \_\_webpack\_require\_\_(moduleId): 手动实现加载一个模块。对已加载过的模块进行缓存，对未加载过的模块，执行 id 定位到\_\_webpack\_modules\_\_中的包裹函数，执行并返回 module.exports，并缓存

\_\_webpack\_require\_\_(0): 运行第一个模块，即运行入口模块3. 涉及到多个 chunk 的打包方式中，比如 code spliting，webpack 中会有 jsonp 加载 chunk 的运行时代码。

四十、打包器(webpack/rollup) 如何将打包后的 js 资源注入 html 中

html-webpak-plugin/@rollup/plugin-html而注入的原理为当打包器已生成 entryPoint 文件资源后，获得其文件名及 publicPath并将其注入到 html 中，它在compilation处理资源的processAssets获得其打包生成的资源

四十一、webpack5 模块联邦

四十二、webpack的proxy原理？为什么能解决跨域？

proxy工作原理实质上是利用http-proxy-middleware这个http代理中间件实现请求转发给其他服务器

在开发阶段webpack-dev-server会启动一个本地开发服务器，通过设置webpack proxy实现代理请求后，相当于浏览器与服务端中添加一个代理者，当本地发送请求的时候，代理服务器响应该请求，并将请求转发到目标服务器，目标服务器响应数据后再将数据返回给代理服务器，最终再由代理服务器将数据响应给本，在代理服务器传递数据给本地浏览器的过程中，两者同源，并不存在跨域行为，这时候浏览器就能正常接收数据

四十三、Vite 和Webpack的区别

1. 启动方式不一样。vite在启动的时候不需要打包，所以不用分析模块与模块之间的依赖关系，不用进行编译。这种方式就类似于我们在使用某个UI框架的时候，可以对其进行按需加载。同样的，vite也是这种机制，当浏览器请求某个模块时，再根据需要对模块内容进行编译。按需动态编译可以缩减编译时间，当项目越复杂，模块越多的情况下，vite明显优于webpack 2. 热更新方面，效率更高。当改动了某个模块的时候，也只用让浏览器重新请求该模块，不需要像webpack那样将模块以及模块依赖的模块全部编译一次。3. vite相关生态没有webpack完善，vite可以作为开发的辅助。

四十四、js 代码压缩 minify 的原理是什么

通过 AST 分析，根据选项配置一些策略，来生成一颗更小体积的 AST 并生成代码。目前前端工程化中使用 terser (opens new window)和 swc (opens new window)进行 JS 代码压缩，他们拥有相同的 API。

常见用以压缩 AST 的几种方案如下:

1. 去除多余字符: 空格，换行及注释，一般来说中文会占用更大的空间，多余的空白字符会占用大量的体积，如空格，换行符，另外注释也会占用文件体积。当把所有空白符合注释都去掉之后代码体积得到减少

2. 压缩变量名，变量名函数名及属性名缩短变量的命名也需要 AST 支持不至于在作用域中造成命名冲突。

3. 解析程序逻辑: 编译预计算

HTTP相关的面试题

一、get和post之间的区别

1. GET方法的参数应该放在url中，POST方法参数应该放在body中

2. 两者都是使用HTTP协议，在网络上是`明文`传输的，只要在网络节点上抓包，就能完整地获取数据报文，所以从传输的角度来说get和post都是不安全的，post比get更安全的地方在于post的参数不会被保存在浏览器历史或web服务器日志中

3. （大多数）浏览器通常都会限制url长度在2K个字节，而（大多数）服务器最多处理64K大小的url

4. GET用于获取信息，是无副作用的，是幂等的，且可缓存；POST用于修改服务器上的数据，有副作用，非幂等，不可自动缓存

二、从 URL 输入到页面展现到底发生什么？

1. 用户输入的是关键字会交给浏览器的默认搜索引擎生产URL，如果是url客户端会进行DNS解析将域名解析成IP地址 2. 客户端利用TCP协议与服务器端进行三次握手连接

3. 握手结束之后客户端会向客户端开始发送 HTTP 请求报文4. 服务器端处理请求并返回 HTTP 报文

5. 客户端会根据返回的信息解析渲染页面 6. 客户端利用TCP协议与服务器端四次挥手断开连接

三、DNS如何将域名解析成IP地址？（浏览器如何通过域名去查询URL对应的IP呢？）

首先搜索浏览器的 DNS 缓存，缓存中维护一张域名与 IP 地址的对应表，若没有命中则继续搜索操作系统的 DNS 缓存，若仍然没有命中，则操作系统将域名发送至本地域名服务器，本地域名服务器采用递归查询自己的 DNS 缓存，查找成功则返回结果，若本地域名服务器的 DNS 缓存没有命中，则本地域名服务器向上级域名服务器进行迭代查询，首先本地域名服务器向根域名服务器发起请求，根域名服务器返回顶级域名服务器的地址给本地服务器，本地域名服务器拿到这个顶级域名服务器的地址后，就向其发起请求，获取权限域名服务器的地址，本地域名服务器根据权限域名服务器的地址向其发起请求，最终得到该域名对应的 IP 地址，本地域名服务器将得到的 IP 地址返回给操作系统，同时自己将 IP 地址缓存起来，操作系统将 IP 地址返回给浏览器，同时自己也将 IP 地址缓存起，至此，浏览器就得到了域名对应的 IP 地址并将 IP 地址缓存起。

四、TCP如何进行三次握手？

刚开始客户端处于 closed 的状态，服务端处于 listen 状态。然后

1. 第一次握手：客户端给服务端发一个 SYN 报文，并指明客户端的初始化序列号 ISN(c)。此时客户端处于 SYN\_Send 状态

2. 第二次握手：服务器收到客户端的 SYN 报文之后，会以自己的 SYN 报文作为应答，并且也是指定了自己的初始化序列号 ISN(s)，同时会把客户端的 ISN + 1 作为 ACK 的值，表示自己已经收到了客户端的 SYN，此时服务器处于 SYN\_REVD 的状态

3. 第三次握手：客户端收到 SYN 报文之后，会发送一个 ACK 报文，当然，也是一样把服务器的 ISN + 1 作为 ACK 的值，表示已经收到了服务端的 SYN 报文，此时客户端处于 establised 状态

4. 服务器收到 ACK 报文之后，也处于 establised 状态，此时，双方以建立起了链接

五、为啥只有三次握手才能确认双方的接受与发送能力是否正常，而两次却不可以？

第一次握手：客户端发送网络包，服务端收到了。这样服务端就能得出结论：客户端的发送能力、服务端的接收能力是正常的

第二次握手：服务端发包，客户端收到了。这样客户端就能得出结论：服务端的接收、发送能力，客户端的接收、发送能力是正常的。不过此时服务器并不能确认客户端的接收能力是否正常

第三次握手：客户端发包，服务端收到了。这样服务端就能得出结论：客户端的接收、发送能力正常，服务器自己的发送、接收能力也正常

六、（ISN）是固定的吗？

不是，是动态生成的，三次握手是为了方便让对方知道接下来接收数据的时候如何按序列号组装数据，如果ISN是固定的，攻击者很容易猜出后续的确认号，有安全问题

七、什么是半连接队列？

服务器第一次收到客户端的 SYN 之后，就会处于 SYN\_RCVD 状态，此时双方还没有完全建立其连接，服务器会把此种状态下请求连接放在一个队列里，我们把这种队列称之为半连接队列。当然还有一个全连接队列是已经完成三次握手建立起连接的就会放在全连接队列中。如果队列满了就有可能会出现丢包现象。

八、SYN-ACK 重传次数的问题？

服务器发送完SYN－ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传，如果重传次数超 过系统规定的最大重传次数，系统将该连接信息从半连接队列中删除。注意，每次重传等待的时间不一定相同，一般会是指数增长，例如间隔时间为 1s, 2s, 4s, 8s, ....

九、三次握手过程中可以携带数据吗？

第三次握手的时候，是可以携带数据的。第一次、第二次握手不可以携带数据，因为此时客户端已经处于 established 状态对于客户端来说，他已经建立起连接了，并且也已经知道服务器的接收、发送能力是正常的了，所以能携带数据页没啥毛病

十、TCP三次握手的作用?

1、确认双方的接受能力、发送能力是否正常。2、指定自己的初始化序列号，为后面的可靠传送做准备。

3、如果是 https 协议的话，三次握手这个过程，还会进行数字证书的验证以及加密密钥的生成到。

4、为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误

十一、 HTTP请求1.1当中请求方法有多少种？

请求方法包含8种：GET、POST、PUT、DELETE、PATCH、HEAD、OPTIONS、TRACE

十二、服务器如何处理请求并返回HTTP报文？

webServer对于不同用户发送的请求，会结合配置文件，把不同请求委托给服务器上处理相应请求的程序进行处理然后返回后台程序处理产生的结果作为响应，首先浏览器发送过来的请求先经过控制器，控制器进行逻辑处理和请求分发，接着会调用模型，这一阶段模型会获取 redis db 以及 MySQL 的数据，获取数据后将渲染好的页面，响应信息会以响应报文的形式返回给客户端，最后浏览器通过渲染引擎将网页呈现在用户面前

十三、浏览器如何解析渲染页面？

1. 根据HTML解析出DOM树，DOM树解析的过程是一个深度优先遍历。即先构建当前节点的所有子节点，再构建下一个兄弟节点。在读取 HTML 文档，构建 DOM 树的过程中，若遇到 script 标签，则 DOM 树的构建会暂停，直至脚本执行完毕

2. 根据 CSS 解析生成 CSS 规则树，解析 CSS 规则树时 js 执行将暂停，直至 CSS 规则树就绪，浏览器在 CSS 规则树生成之前不会进行渲染

3. 结合 DOM 树和 CSS 规则树，生成渲染树，DOM 树和 CSS 规则树全部准备好了以后，浏览器才会开始构建渲染树。精简 CSS 并可以加快 CSS 规则树的构建，从而加快页面相应速度。

4. 根据渲染树计算每一个节点的信息(布局)，通过渲染树中渲染对象的信息，计算出每一个渲染对象的位置和尺寸，在布局完成后，发现了某个部分发生了变化影响了布局，那就需要倒回去重新渲染（回流）

5. 根据计算好的信息绘制页面，绘制阶段，系统会遍历呈现树，并调用呈现器的“paint”方法，将呈现器的内容显示在屏幕上；某个元素的背景颜色，文字颜色等，不影响元素周围或内部布局的属性，将只会引起浏览器的重绘；某个元素的尺寸发生了变化，则需重新计算渲染树，重新渲染引发回流；

十四、TCP如何进行四次挥手？

刚开始双方都处于 establised 状态，假如是客户端先发起关闭请求，则：

1. 第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于FIN\_WAIT1状态。

2. 第二次握手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 + 1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 CLOSE\_WAIT状态。

3. 第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 LAST\_ACK 的状态。

4. 第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 + 1 作为自己 ACK 报文的序列号值，此时客户端处于 TIME\_WAIT 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态

5. 服务端收到 ACK 报文之后，就处于关闭连接了，处于 CLOSED 状态。

十五、四次握手中TIME\_WAIT，为什么客户端发送 ACK 之后不直接关闭，而是要等一阵子才关闭?

确保服务器是否已经收到了我们的 ACK 报文，如果没有收到的话，服务器会重新发 FIN 报文给客户端，客户端再次收到 ACK 报文之后，就知道之前的 ACK 报文丢失了，然后再次发送 ACK 报文。至于 TIME\_WAIT 持续的时间至少是一个报文的来回时间。一般会设置一个计时，如果过了这个计时没有再次收到 FIN 报文，则代表对方成功就是 ACK 报文，此时处于 CLOSED 状态

十六、四次握手状态码都分别有什么用处？

LISTEN - 侦听来自远方TCP端口的连接请求；SYN-SENT - 在发送连接请求后等待匹配的连接请求；

SYN-RECEIVED - 在收到和发送一个连接请求后等待对连接请求的确认；

ESTABLISHED - 代表一个打开的连接，数据可以传送给用户；

FIN-WAIT-1 - 等待远程TCP的连接中断请求，或先前的连接中断请求的确认；

FIN-WAIT-2 - 从远程TCP等待连接中断请求；CLOSE-WAIT - 等待从本地用户发来的连接中断请求；

CLOSING -等待远程TCP对连接中断的确认；LAST-ACK -等待原来发向远程TCP的连接中断请求的确认

TIME-WAIT -等待足够的时间以确保远程TCP接收到连接中断请求的确认；CLOSED - 没有任何连接状态；

十七、为什么JS会阻塞页面加载？

js会阻塞DOM的解析，所以会阻塞页面的加载，由于js会操控css和dom，如果在修改这些属性时渲染界面那么渲染进程前后拿到的元素数据就可能不一致了。为了防止渲染出现不可预期的结果，浏览器设置`GUI渲染引擎和JS引擎为互斥的`，当执行js的时候，GUI线程就会挂起，GUI更新会被保存在一个队列当中进行等待，直到引擎空闲时才会被重新执行，如果js执行的时间过长，页面加载就会受到影响，出现阻塞渲染的情况，如果js代码没有操作dom的操作，那么则应该异步加载。

十八、CSS加载会造成阻塞吗

CSS不会阻塞DOM的解析会阻塞DOM的渲染，CSS不会阻塞js文件的下载，但是会阻塞js的执行

CSSOM（提供js操作样式表的能力；为布局树的合成提供基础的样式信息）和DOM进程是并行构建，所以css加载不会阻塞DOM的解析；Render Tree是依赖于CSS Tree和DOM Tree的，所以会等待两棵树都构建完毕之后才会进行渲染，所以css会阻塞DOM的渲染；为了防止渲染出现不可预期的结果，浏览器设置`GUI渲染引擎和JS引擎为互斥的，有些时候js的执行会依赖前面css所控制的数据，浏览器也无法感知脚本内容到底是什么，为避免样式获取，因此只能等到前面的样式下载完毕在执行js，所以`CSS不会阻塞js文件的下载，但是会阻塞js的执行`

十九、defer 和 async 的区别 ?

1. 两者都是在DOM渲染结束后进行加载，不会阻塞DOM

2. async在外部js加载完后浏览器空闲时Load事件触发执行前执行不能保证先后顺序对内联脚本无作用

defer是在外部js加载完后整个文档解析完成，触发DOMContentLoaded之前进行执行，

二十、DOMContentLoaded 与 load 的区别 ?

DOMContentLoaded是在dom解析完毕之后触发，不包括样式图片等资源，onLoad事件是在样式，dom脚本，图片等所有资源已经加载完毕之后触发也就是优先级 DOMContentLoaded > onLoad

async脚本是在load事件之前触发，那就有可能是在DOMContentLoaded事件之前或者之后执行

情况1： HTML 还没有被解析完的时候，async脚本已经加载完了，那么 HTML 停止解析，去执行脚本，脚本执行完毕后触发DOMContentLoaded事件

情况2： HTML 解析完了之后，async脚本才加载完，然后再执行脚本，那么在HTML解析完毕、async脚本还没加载完的时候就触发DOMContentLoaded事件

defer脚本那么这一块脚本将不会影响 HTML 文档的解析，而是等到HTML 解析完成后才会执行。而 DOMContentLoaded 只有在 defer 脚本执行结束后才会被触发。

情况1：HTML还没解析完成时，defer脚本已经加载完毕，那么defer脚本将等待HTML解析完成后再执行。defer脚本执行完毕后触发DOMContentLoaded事件

情况2：HTML解析完成时，defer脚本还没加载完毕，那么defer脚本继续加载，加载完成后直接执行，执行完毕后触发DOMContentLoaded事件

二十一、为什么CSS动画比JavaScript高效 - 避免DOM的重绘和重排更高效

二十二、谈一谈你对requestAnimationFrame（rAF）理解

高性能动画：动画帧率作为衡量标准，一般画面在60fps的帧率下面是最流畅的换算一下就是，每一帧要在 16.7ms 内完成渲染；rAF告诉浏览器执行动画并在下一次浏览器重绘的时候执行相应的函数来更新动画，该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

当我们调用这个函数的时候，我们告诉它需要做两件事：我们需要新的一帧和当你渲染新的一帧时需要执行我传给你的回调函数

附加题：rAF优势

1. rAF是由系统来决定回调函数的执行时机，rAF 的执行步伐跟着系统的绘制频率走，它能保证回调函数在屏幕每一次的绘制间隔中只被执行一次(上一个知识点刚刚梳理完「函数节流」)，这样就不会引起丢帧现象，也不会导致动画出现卡顿的问题。
2. 自动调节频率如果callback工作太多无法在一帧内完成会自动降低为30fps虽然降低了但是总比掉帧好。

附加题：与setTimeout动画对比的话，有以下几点优势

setTimeout当页面隐藏或者最小化时仍然在后台执行动画浪费CPU。rAF不一样，当页面处理未激活的状态时，该页面的屏幕绘制任务也会被系统暂停，因此跟着系统步伐走的rAF也会停止渲染，当页面被激活时，动画就从上次停留的地方继续执行，有效节省了 CPU 开销。

附加题：rAF调用时机 - 在重新渲染前调用、很可能在宏任务之后不去调用

根据经验，如果 JavaScript 方法需要绘制或者直接改变属性，我会选择 requestAnimationFrame，只要涉及到重新计算元素位置，就可以使用它。

涉及到 AJAX 请求，添加/移除 class （可以触发 CSS 动画），我会选择 \_.debounce 或者 \_.throttle ，可以设置更低的执行频率（例子中的200ms 换成16ms）。

二十三、常见的浏览器内核有哪些？

浏览器 | 内核（渲染引擎）| javasrcipt引擎

chrome | webkit | V8

safari | webkit | JavaSrciptCore

FireFox | Gecko | SpiderMonkey

IE | Trident | JScript（IE3.0-IE8.0）

Edge | EdgeHTML | Chakra(for JavaScript)

Opera | Presto | Linear/ Futhark / Carakan

二十四、浏览器的主要组成部分（7个）?

1. 【用户界面】- 包括地址栏、前进后退按钮、书签菜单等；
2. 【浏览器引擎】 - 用户界面与呈现引擎之间传送指令；
3. 【呈现引擎】- 负责显示请求的内容。如果请求的内容是HTML，他就负责解析HTML和CSS内容，并将解析后的内容显示在屏幕上；
4. 【网络】- 用于网络调用，比如HTTP请求；
5. 【用户界面后端】- 用于绘制基本的窗口小部件，比如组合框和窗口；
6. 【JavaScript解释器】- 用于解析和执行js代码；
7. 【数据存储】- 持久层，浏览器会在硬盘上面保存各种数据，新的 HTML 规范 (HTML5) 定义了“网络数据库”，这是一个完整（但是轻便）的浏览器内数据库；

值得注意的是：chrome浏览器每一个标签页都对应独立的呈现引擎实例，每一个标签页都是一个独立进程。

二十四、浏览器进程和线程 - 进程是操作系统资源分配的基本单位，一个进程就是一个程序的运行实例，进程当中包括线程；启动一个程序时操作系统会为该程序创建一块内存用来存放内存、运行数据和一个执行任务主线程，我们把这样的一个运行环境叫做进程，进程中使用多线程并行处理能提高运算效率

附加题：线程是由进程来启动和管理的，线程是不允许单独存在的，为了提示浏览器的稳定性和安全性，浏览器采用了多进程模型

二十五、进程和线程之间的关系

1. 进程中的任一线程执行出错，都会导致整个进程的崩溃

2. 线程之间共享进程中的数据，线程之间可以对进程的公共数据进行读写操作

3. 当一个进程关闭之后，操作系统会回收进程所占用的内存；即使进程中任意线程因为操作不当导致内存泄漏，当进程退出时这些内存也会被回收

4. 进程之间的内容是相互隔离的，由于这个特性，即使一个进程崩溃也不会影响其他进程，如果进程之间需要进行数据的通信，就需要使用进程间通信的\*\*IPC\*\*机制

二十六、浏览器当中的5个进程

1. 渲染进程：每个显卡都有单独的渲染进程，核心用于渲染页面
2. 插件进程： chrome安装一些插件
3. 网络进程：主要处理网络资源加载(html,css,js)
4. 浏览器进程： 负责页面显示，用户交互，子进程管理，提供存储等等
5. GPU进程： 3d绘制,提高性能

二十七、DOM如何生成的

1. 当服务端返回的类型是text/html时浏览器会将收到的数据通过HTMLParser进行解析 (边下载边解析)

2. 在解析前会执行预解析操作，会预先加载JS、CSS等文件

3. 字节流 -> 分词器 -> Tokens -> 根据token生成节点 -> 插入到 DOM树中

4. 遇到js:在解析过程中遇到script标签，HTMLParser会停止解析，（下载）执行对应的脚本。

5. 在js执行前，需要等待当前脚本之上的所有CSS加载解析完毕（js是依赖css的加载）

6. CSS样式文件尽量放在页面头部，CSS加载不会阻塞DOM tree解析,浏览器会用解析出的DOM TREE和 CSSOM 进行渲染，不会出现闪烁问题。如果CSS放在底部，浏览是边解析边渲染，渲染出的结果不包含样式，后续会发生重绘操作

7. JS文件放在HTML底部，防止JS的加载、解析、执行堵塞页面后续的正常渲染

二十八、重绘和重排（回流）

重排：当渲染树的一部分必须更新并且节点的尺寸发生了变化，浏览器会使渲染树中受到影响的部分失效，并重新构造渲染树。 主要行为表现为：添加和删除可见的dom、元素的位置改变、元素的尺寸改变（外边距、内边距、边框厚度、宽高等几何属性）、页面渲染初始化以及浏览器窗口尺寸改变

重绘：是在一个元素的外观被改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

附加题：重绘不一定需要重排，重排必然会导致重绘

二十九、减少reflow、repaint

1. 不要一条一条的修改DOM的样式，可以先定义好css的class，然后修改DOM的className。

2. 不要把DOM结点的属性值放在一个循环里当成循环里的变量。

3. 为动画的HTML元件使用fixed或absolute的position，那么修改他们的css是不会reflow

三十、打开一个网页，有几个进程

1 个网络进程、1 个浏览器进程、1 个 GPU 进程以及 1 个渲染进程。如果打开的页面有运行插件的话，还需要再加上 1 个插件进程。

三十一、COOKIE和SESSION有什么区别？

客户端与服务器端进行会话的时候需要使用客户端的唯一标识确定身份维持会话由于http协议是无状态协议，所以服务器需要记录用户的状态的时候就需要使用session机制来识别具体的用户；每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端，服务端会在HTTP协议中告诉客户端需要在Cookie里面记录一个Session ID以后每次请求把这个会话ID发送到服务器所以Cookie是用来实现Session跟踪

三十二、Cookie、localStorage、sessionStorage

1. cookie是为了解决客户端与服务器端的会话问题、存放在本地浏览器关闭了也依然有效；浏览器发起一个http请求，服务器相应请求后会带回一个set-cookie的字段，浏览器会保存cookie每一次向服务器发送请求时都会带上这个cookie，cookie其实就是存储在浏览器端的一个小的文本文件，内部以键值对的方式进行存储
2. localStorage适合持久化浏览器缓存数据
3. sessionStorage适合一次性浏览器临时数据保存，存储本次浏览信息记录，这样子页面关闭的话，就不需要这些记录了，还有对表单信息进行维护，这样子页面刷新的话，也不会让表单信息丢失。

三十三、cookie的缺点

1. 容量缺陷：cookie的大小上限是4KB，只能存储少量的信息
2. 降低性能：cookie的同源策略导致同站点的请求无论是否需要cookie，在请求发送的时候都会带上完整的cookie信息，造成了资源浪费
3. 安全缺陷：cookie作为文本在客服端与服务器端来回传递，很容易被窃取，即使设置了HTTPOnly，仍然可以通过js脚本获取到cookie值不安全

三十四、浏览器缓存

浏览器先向代理服务器发起Web请求，再将请求转发到源服务器。它属于共享缓存，所以很多地方都可以使用其缓存资源，因此对于节省流量有很大作用。浏览器缓存是将文件保存在客户端，在同一个会话过程中会检查缓存的副本是否足够新，在后退网页时，访问过的资源可以从浏览器缓存中拿出使用。通过减少服务器处理请求的数量，用户将获得更快的体验。

附加题：浏览器缓存机制

浏览器与服务器通信的方式为应答模式，那么浏览器第一次向服务器发起该请求后拿到请求结果，会根据响应报文中HTTP头的缓存标识，决定是否缓存结果，是则将请求结果和缓存标识存入浏览器缓存中

三十五、强缓存、协商缓存

1. 强缓存：http请求的请求头设置了Expires或者Cache-Control，那么资源在客户端缓存的有效期内再次请求就会命中强缓存浏览器便不会向服务器发送请求, 而是直接读取缓存
2. 协商缓存：http请求的请求头设置了Last-Modified，If-Modified-Since或ETag、If-None-Match，当浏览器对某个资源的请求没有命中强缓存，就会发一个请求到服务器，验证协商缓存是否命中，如果协商缓存命中，请求响应返回的http状态为304并且会显示一个Not Modified的字符串,查看单个请求的Response Header，也能看到304的状态码和Not Modified的字符串，只要看到这个就可说明这个资源是命中了协商缓存，然后从客户端缓存中加载的，而不是服务器最新的资源

附加题：共同点命中从客户端缓存中加载资源 区别 强缓存不发请求到服务器协商缓存会发请求到服务器

附加题：

三十六、跟cookie相关的请求头的含义

1. Expires：Expires缺省是会话性cookie，保存在客户端内存当中，当浏览器关闭时，cookie就会失效；2. Expires设置成session时是持久性保存在硬盘上即使浏览器关闭了依然有效直至过期或者清除 Cookie

3.Max-Age：表示距离cookie失效需要经过n秒，优先级高于Expires

4. Domain： 指定了cookie的主机名，不能跨域，默认情况下是当前文档访问地址中的主机部分（但是不包含子域名）

5. Path：Path 指定了一个 URL 路径，这个路径必须出现在要请求的资源的路径中才可以发送 Cookie 首部。比如设置 Path=/docs，/docs/Web/ 下的资源会带 Cookie 首部，/test 则不会携带 Cookie 首部。

6. Secure属性：标记为 Secure 的 Cookie 只应通过被HTTPS协议加密过的请求发送给服务端。使用 HTTPS 安全协议，可以保护 Cookie 在浏览器和 Web 服务器间的传输过程中不被窃取和篡改。

7. HTTPOnly：可以防止客户端脚本通过 document.cookie 等方式访问有助于避免 XSS 攻击。

8. SameSite：SameSite 属性可以让 Cookie 在跨站请求时不会被发送，从而可以阻止跨站请求伪造攻击（CSRF）。设置了这个属性，iframe，post，img，ajax都不会携带cookie，而get，<a>,<link>仍然会携带cookie

三十七、浏览器与服务器建立了一个 TCP 连接后会在一个 HTTP 请求完成后断开？哪种情况会断开？

不会断开，早在http/1.0中一个服务器发送完HTTP请求后，TCP就会断开链接。但是每次请求都会重新建立和断开TCP连接，代价太大了。http/1.1将Connection头写进标准，并且默认保持开启持久化连接，除非请求中声明Connection: close 否则浏览器和服务器之间是会持续一段时间的TCP连接不会一个请求就结束掉。

答： 默认情况下建立的TCP连接不会自动断开，除非将请求头中的Connection设置成close状态才会才请求完成后关闭连接

三十八、一个 TCP 连接可以对应几个 HTTP 请求？ - 如果是持久连接的情况下，一对多

三十九、一个 TCP 连接中 HTTP 请求发送可以一起发送么？比如一起发三个请求，再三个响应一起接收答： http/1.1协议存在Pipelining技术可以完成这个多个请求同时发送，但是由于浏览器默认是关闭的，所以认为这并不是可行的。在http2.0提供了Multiplexing，多个HTTP请求可以在同一个`TCP`连接中并行进行。

附加题: 为什么1.1的Pipelining要求发送的和请求的顺序发送响应

由于1.1 是个文本协议，同时返回的内容也并不能区分对应于哪个发送的请求所以顺序必须维持一致。

附加题：Pipelining缺点：一些代理服务器不能正确的处理Pipelining；正确的流水线实现是复杂的。

四十、在 HTTP/1.1 时代，浏览器是如何提高页面加载效率的呢？

1. 维持和服务器已经建立的 `TCP` 连接，在同一连接上顺序处理多个请求。

2. 和服务器建立多个 `TCP` 连接。

四十一、为什么有的时候刷新页面不需要重新建立 SSL 连接？

TCP连接有的时候会被浏览器和服务端维持一段时间。TCP不需要重新建立SSL自然也会用之前的。

四十二、浏览器对同一Host建立TCP连接到数量有没有限制？

答案：有。Chrome最多允许对同一个Host建立六个TCP连接。不同的浏览器有一些区别；

四十三、收到的 HTML 如果包含几十个图片标签，这些图片是以什么方式、什么顺序、建立了多少连接、使用什么协议被下载下来的呢

答案：如果图片都是HTTPS连接并且在同一个域名下，那么浏览器在SSL握手之后就会跟服务器商量能不能使用HTTP2.0，如果能的话就是用Multiplexing功能在这个连接上进行多路传输。不过也未必会所有挂在这个域名的资源都会使用一个TCP连接去获取，但是可以确定的是 Multiplexing很可能会被用到。

如果只能使用http/1.1，那浏览器就会在一个host上建立多个TCP连接，连接的数量的最大限制取决于浏览器设置，这些连接会在空闲的时候被浏览器用来发送新的请求。

四十四、TCP的流量控制机制

为了让发送方发送数据的速度不要太快，要让接收方来得及接收。TCP采用大小可变的滑动窗口进行流量控制窗口大小（每次传输的数据大小）的单位是字节

+ 当一个连接建立时，连接的每一端分配一个缓冲区来保存输入的数据，并将缓冲区的大小发送给另一端。

+ 当数据到达时，接收方发送确认，其中包含了自己剩余的缓冲区大小。

+ 如果接收方应用程序读数据速度与数据到达速度一样快，接收方将在每一确认中发送一个正的窗口通告。

+ 如果发送方操作的速度快于接收方，接收到的数据最终将充满接收方的缓冲区，导致接收方通告一个零窗口 。发送方收到一个零窗口通告时，必须停止发送，直到接收方重新通告一个正的窗口。

四十五、TCP和UDP最大的区别是什么?

UDP：无连接，不可靠传输，不使用流量控制和拥塞控制，支持一对一，一对多，多对一和多对多交互通信，采用面向报文的传输方式且首部开销小，仅8字节，适用于实时应用，例如视频会议、直播

TCP：面向连接，可靠传输（数据顺序和正确性），使用流量控制和拥塞控制，只能是一对一通信，采用面向字节流的传输方式且首部最小20字节，最大60字节，适用于要求可靠传输的应用，例如文件传输

四十六、HTTP发展历程

1.HTTP/0.9 在传输过程中没有请求头和请求体，服务器响应没有返回头信息，内容采用ASCII字符流来进行传输 HTML 2. HTTP/1.0 增加了请求头和响应头，实现多类型数据传输 3. HTTP/1.1 默认开启持久链接，一个TCP链接上可以传输多个HTTP请求，采用管线化方式（每个域名最多维护6个TCP持久链接）解决队头阻塞问题 （服务端需要按顺序依次处理请求）在同一个TCP连接当中数据通信时是按照次序。完美支持数据分块传输并引入客户端cookie机制、安全机制4. HTTP/2.0 解决网络带宽使用率低 （TCP慢启动，多个TCP竞争带宽，队头阻塞）采用多路复用机制（一个域名使用一个TCP长链接，通过二进制分帧层来实现。只需要一个连接即可实现并行，头部压缩降低开销及服务端推送 5. HTTP/3.0 解决TCP队头阻塞问题采用QUIC协议。QUIC协议是基于UDP的

四十七、HTTP常见的状态码

http状态码的作用是服务器告诉客户端当前请求响应的状态，通过状态码就能判断和分析服务器的运行状态1开头表示消息2表示成功3 表示重定向4 表示请求错误5 表示服务器错误

100:临时响应需要继续发送请求101切换协议-websocket或http2升级200成功201已创建请求成功并且创建了新的资源 202已创建，接收尚未处理203返回信息可能来自另一来源204无内容返回205重置内容，无内容返回206部分内容，服务器成功处理部分请求-断点请求300针对请求，服务器可执行多种操作301网页自动已永久移动到新位置302临时重定向不会缓存303针对不同的位置使用各自GET请求来检索响应304协商缓存305应该使用代理访问请求的网页307 临时重定向400服务器不理解请求语法401未授权403拒绝请求404未找到405方法禁用406不接受407需要代理授权408请求超时500服务器内部错误501不具备完成请求的功能502错误网关503服务不可用504网关超时505HTTP版本不支持

四十八、同样是重定向，307，303，302的区别？

302是http1.0协议状态码在http1.1细化302状态码⼜:303和307，303明确表示应当采⽤get⽅法获取资源会把POST->GET请求进⾏重定向307会遵照浏览器标准不会从post变为get

四十九、302、304、301区别

301永久重定向缓存 302 临时重定向无缓存常用未登录304 协商缓存客户端有缓存要使用客户端的缓存有请求头无内容

五十、HTTP常见的请求头以及作用

Accept能够接受的回应内容类型Accept-Charset能够接受的字符集Accept-Encoding能够接受的编码方式列表Accept-Language内容en-US Authorization用于超文本传输协议认证信息Cache-Control指定缓存机制Connection连接方式keep-alive；Cookie； Content-Length请求体长度Content-Type请求体多媒体类型 Content-Type: application/x-www-form-urlencoded；Date发送该消息日期和时间 Host服务器域名和协议端口号

五十一、HTTPS和HTTP区别

1. HTTPS是HTTP协议的安全版本，HTTP协议的数据传输是明文是不安全，HTTPS使用了SSL/TLS协议进行了加密处理，相对更安全
2. HTTP 和 HTTPS 使用连接方式不同，默认端口也不一样，HTTP是80，HTTPS是443
3. HTTPS 由于需要设计加密以及多次握手，性能方面不如 HTTP
4. HTTPS需要SSL，SSL 证书需要钱，功能越强大的证书费用越高

五十二、HTTPS与服务器连接的流程

1. 建立SSL连接 2. 接收请求后会将网站支持的证书信息（证书中包含公钥）传送一份给客户端3. 服务器开始协商SSL连接安全等级即信息加密等级4. 客户端建立会话密钥利用网站的公钥将会话密钥加密并传送给网站 5. 服务器利用自己的私钥解密出会话密钥6. 服务器利用会话密钥加密与客户端之间的通信

五十三、为什么说HTTPS比HTTP安全? HTTPS是如何保证安全的？

HTTPS与HTTP虽然只差一个SSL但是通信安全得到了大大的保障，通过摘要算法保障通信的完整性，通过混合算法确保通信的机密性，通过数字签名确保通信的身份认证和不可否定性，同时引入第三方证书机构，确保公开秘钥的安全性

五十四、HTTPS是如何进行加密的？使用的什么加密算法？

采用的是对称加密+非对称加密，也就是混合加密；对称加密是加密和解密使用的秘钥都是同一个，保证了密钥的安全使通信具备机密性，采用非对称加密解决秘钥交换的问题发送密文的一方使用对方的公钥进行加密处理“对称的密钥”，然后对方用自己的私钥解密拿到对称的密钥；通过摘要算法即散列函数哈希函数将数据进行压缩附在原文后，服务器端收到后进行对比来确保数据并没有被篡改；最后通过私钥加密公钥解密的数字签名来确认消息的发送方；并引入第三方证书机构来确保公开密钥的安全性

五十五、http2.0 为什么较http1.1速度更快

主要是因为http2.0 在解析html过程当中会主动推送静态资源至客户端并且放在缓存当中，当静态资源进行请求的时候直接命中缓存就可以了，所以速度会相当快

五十六、HTTP/2.0 都有哪些特性？

HTTP/2.0采用二进制格式而非文本格式，通过多路复用机制而非有序并阻塞的、只需一个连接即可实现并行解决网络带宽使用率低 TCP慢启动，多个TCP竞争带宽，队头阻塞；通过二进制分帧层来实现一个域名使用一个TCP长链接以及报头压缩降低开销和服务端推送

五十七、http2.0 头部压缩原理

头部压缩使用的是HPACK算法，在客户端和服务器两端建立字典，用索引号表示重复的字符串，采用哈夫曼编码来压缩整数和字符串，可以达到50%~90%的高压缩率。1. 在客户端和服务器端使用“首部表”来跟踪和存储之前发送的键值对，对于相同的数据，不再通过每次请求和响应发送；2. 首部表在HTTP/2的连接存续期内始终存在，由客户端和服务器共同渐进地更新；3. 每个新的首部键值对要么被追加到当前表的末尾，要么替换表中之前的值。

五十八、OSI七层模型

应用层、表示层、会话层、传输层、网络层、数据链路层、物理层

应用层：定义了应用进程之间的交互规则，此时的数据叫做报文

表示层：进行解释交换数据，进行数据压缩，数据加密以及数据描述，使数据不具备格式差异

会话层：建立、管理和终止表示层实体之间的通信会话，数据交换的定界和同步功能

传输层：通信提供服务，处理数据包错误、数据包次序的其他一些关键传输问题，传输协议是TCP和UDP

网络层：IP层，选择合适的网间路由和交换节点，确保数据按时成功传送，将数据进行分包

数据链路层：两台主机之间的数据传输，将IP数据组装成帧

物理层：实现计算机节点之间比特流的透明传送

五十九、TCP 的 Keepalive 和 HTTP 的 Keep-Alive 是一个东西吗

这两个完全是两样不同东西，实现的层面也不同：HTTP -Keep-Alive是由应用层实现的称为 HTTP 长连接；TCP -Keepalive，是由TCP层实现的是TCP 保活机制；HTTP 的 Keep-Alive建立HTTP 长连接，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态。长连接不仅仅减少了 TCP 连接资源的开销；TCP的Keepalive是定义一个时间段，在这个时间段内，如果没有任何连接相关的活动，TCP保活机制会开始作用每隔一个时间间隔发送一个探测报文，如果连续几个探测报文都没有得到响应，则认为当前TCP连接已经死亡，系统内核将错误信息通知给上层应用程序

六十、各种缓存的优先级, memory disk http2 push? - emory内存中的缓存 > Disk > Push

六十一、Web 漏洞包含哪一些主流的漏洞类型

1. 注入： SQL注入，命令注入以及LDAP注入

2. 失效的身份认证和会话管理，如果攻击者破解密码窃取密码、会话令牌或者其他漏洞去冒充他人的身份

3. 跨站脚本（XSS）：XSS允许攻击者在受害者的浏览器执行恶意脚本，从而劫持用户会话、钓鱼欺骗等

4. 失效的访问控制： 比如越权访问其他用户的个人资料、查看敏感文件、篡改数据等

5. 安全配置错误：比如服务器的不安全配置，导致敏感信息泄露

6. 敏感信息泄露：比如账号密码未加密存储、敏感数据传输时未加密保护，最终造成数据泄露

7. 攻击检测与防护不足：比如 WAF、主机入侵检测等防御系统部署不全，这块偏向漏洞防御本身。

8. 跨站请求伪造（CSRF）攻击者诱使其他登录用户访问恶意站点，以冒用对方的身份执行一些敏感操作。

9. 使用含有已知漏洞的组件：比如一些第三方的开源库、框架等，尤其是已公开漏洞的旧版本，比如名燥一声的 Struts2 漏洞，因频繁出现漏洞被许多开发者弃用

10. 未受有效保护的 API：比如浏览器和移动 App 中的 JavaScript API，常常因其提供的特殊功能未受有效保护而被滥用，造成不同等级的危害程度。

六十二、在前端开发中，如何获取浏览器的唯一标识？

根据 canvas 可以获取浏览器指纹信息，绘制 canvas，获取 base64的 dataurl，对dataurl 这个字符串进行 md5 摘要计算，得到指纹信息，若在生产环境使用，可以使用 fingerprintjs2 (opens new window)，根据业务需求，如单设备是否可跨浏览器，以此选择合适的 component

六十三、什么是CSRF？

CSRF 跨站请求伪造，通过恶意引导用户一次点击劫持 cookie 进行攻击。跟跨网站脚本（XSS）相比，XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户网页浏览器的信任。

附加题：怎么样避免CSRF的攻击？

1. 使用 JSON API。CSR攻击时请求体通过 <form> 构建，请求头为 application/www-form-urlencoded。它难以发送 JSON 数据被服务器所理解。

2. 生成一个随机的 token，切勿放在 cookie 中，每次请求手动携带该 token 进行校验。

3. SameSite Cookie。设置为 Lax 或者 Strict，禁止发送第三方 Cookie。

六十四、 XSS

XSS跨站脚本攻击，本质上是黑客通过对网页的HTML注入，篡改了原本服务器发给客户端的数据包，在其中插入了恶意的Script代码插入到网页中，当正常用户浏览该页面时，被嵌入的恶意Script代码就会被执行，从而达到恶意攻击正常用户的目的。

六十五、什么是跨域？如何解决跨域（跨越的解决方案）？- 协议，域名，端口三者有一不一样就是跨域

常见的解决方案如下：1. iframe方案 2. 采用script标签的方案 3.CORS在服务器端设置几个响应头，如 Access-Control-Allow-Origin: \* 4. Reverse Proxy，在 nginx/traefik/haproxy 等反向代理服务器中设置为同一域名 5. 使用JSONP的方式

六十六、JSONP 的原理是什么，如何实现？

只能处理 GET 跨域，JSONP 基于两个原理: 动态创建 script使用 script.src 加载请求跨过跨域和script.src 加载的脚本内容为 JSONP即 PADDING(JSON) 格式，使用 JSONP 跨域同样需要服务端的支持, 在请求头部分会多一个callback=padding， 并且响应数据被 padding 包围，这就是 JSONP

JSONP的数据处理：padding 就是处理数据的函数。我们只需要在前端实现定义好 padding 函数即可window.padding = handleData

项目相关的面试题

一、自研的微前端架构

由于我们的产品是以交付场景为主，所以一定会产生定制化，我们以最简单的开发为原则，将公共的运行环境进行抽离比如登陆页、权限控制、统一的请求体系、公共的utils，然后在模块运行的时候再将这些公共的代码加载进来组合成一个正常的vue可以正常的启动，所以我们将公共的运行环境拆分成了基础部分common以及定制化的custom部分

一个完整的项目是由以下3部分组成的

common运行环境和客户自定义的运行环境，这部分我们可以是在vue-cli3初始化的时候生成