**1.apply,call,bind的区别**

apply: 函数中的 this 替换成参数1, 其余参数放数组中. 直接触发函数

call: 函数中的 this 替换成参数1, 其余参数依次摆放. 直接触发函数

bind: 替换函数中的 this 指向 并 传入其他参数,返回新的函数. 不会直接触发函数

**2. vue的传参方式**

(1)组件间传参

a.父给子传参（父：自定义属性传递变量，子：用Props['父元素自定义属性']接受）

b.子给父传参（子：this.$emit(父元素自定义事件，传递的值) 父：自定义事件用参数去接收传递来 的值）

c.兄弟间传参（用Vuex 代替）

通过给eventbus注册事件，别的组件触发事件，实现通信。

①创建一个eventbus对象，即创造一个vue对象，用作传递信息。

②接收信息的一方将事件通过vm.$on(“事件名”,事件处理函数)添加在eventbus对象上。

③发送数据的一方，通过vm.$emit(“事件名”,传递的数据)触发事件，将数据传递。

(2)路由传参

a.通过params

需要在router的path后跟上参数

{

path: "/two/:id/:data",

name: "two",

component: two

}

跳转时

this.$router.push({

path: `/two/1/456` // 直接把数据拼接在path后面

})

接收时

this.$route.params {id:1, data:456}

b.通过query

跳转时

this.$router.push({

path: `/two`,

query: { id: 1, data: 456 }

});

/two?id=1&data=456

接收时

this.$route.query

params 和 query 都是传递参数的，params不会在url上面出现，并且params参数是路由的一部分，是一定要存在的 , query是我们通常看到的url后面的跟在？后面的显示参数

**3.router-link和router.push实现跳转的原理**

router-link

默认会渲染为 a 标签. 可以通过 tag 属性修改为其他标签 自动为 a 标签添加 click 事件. 然后执行 $router.push() 实现跳转

$router.push 根据路由配置的 mode 确定使用 HTML5History 还是 HashHistory 实现跳转 HTML5History : 调用 window.history.pushState() 跳转

HashHistory : 调用 HashHistory.push() 跳转

**4.组件中的data为什么是函数**

避免组件中的数据互相影响。同一个组件被复用多次会创建多个实例，如果 data 是一个对象的话，这些实例用的是同一个构造函数。为了保证组件的数据独立，要求每个组件都必须通过 data 函数返回一个对象作为组件的状态。

**5.什么是跨域**

http://www.codeboy.com:9999

跨域是由浏览器的同源策略造成的。同源策略：指域名、协议、端口相同；如果违反了同源策略就会造成跨域

跨域的解决

cors 一般都是后端搞定，我们只需要顾虑前端的东西

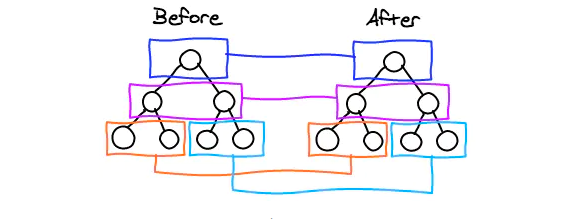
jsonp 了解过，使用DOM操作动态的添加一个元素，而script标签中的src属性是 没有跨域限制的也就是逃避了同源策略，而且只能用get请求

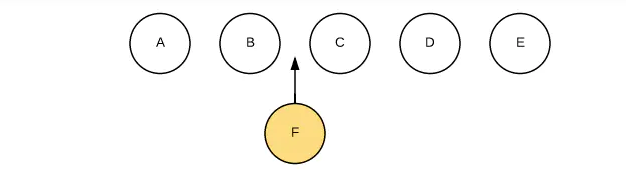
接口代理 通过代理服务器配置来实现，例如Vue中需要在vue.config.js中配置poxy文件

6.**Vue中 key值作用和原理**

在渲染列表的时候使用key，如果不使用的时候会进行警告。当v-for循环渲染时，不加上key的时候会默认采用**原地复用原则**。即不会移动或重新渲染dom元素或者标签，而是单纯的改变数据，对数据进行更新。

key的作用主要是为了高效的更新虚拟DOM。另外vue中在使用相同标签名元素的过渡切换时，也会使用到key属性，其目的也是为了让vue可以区分它们。





**7.cookies，sessionStorage和localStorage的区别？**

共同点：都是保存在浏览器端，且是同源的。

区别：

cookies是为了标识用户身份而存储在用户本地终端上的数据，始终在同源http请求中携带，即cookies在浏览器和服务器间来回传递，而sessionstorage和localstorage不会自动把数据发给服务器，仅在本地保存。

**存储大小的限制不同**。cookie保存的数据很小，不能超过4k，而sessionstorage和localstorage保存的数据大，可达到5M。

**数据的有效期不同**。cookie在设置的cookie过期时间之前一直有效，即使窗口或者浏览器关闭。sessionstorage仅在浏览器窗口关闭之前有效。localstorage始终有效，窗口和浏览器关闭也一直保存，用作长久数据保存。

**作用域不同。**cookie在所有的同源窗口都是共享；sessionstorage不在不同的浏览器共享，即使同一页面；localstorage在所有同源窗口都是共享

**8.webpack是什么，用来做什么的**

webpack是前端构建工具

webpack：是前端模块化方案，侧重模块打包。把开发中所有资源都看成模块，通过配置 loader 和 插件对资源进行处理。实现团队开发项目目录的统一

gulp：强调前端开发的流程，需要进行很多配置。适合打包一些轻量化的任务，比如单独打包css文件

webpack 是把项目当作一个整体，通过一个给定的的主文件，webpack 将从这个文件开始找到你 的项目的所有依赖文件，使用 loaders 处理它们，后打包成一个或多个浏览器可识别的 js 文件

**9.webpack中常见的loader有哪些**

style-loader 将css添加到DOM的内联样式标签style里

css-loader 允许将css文件通过require的方式引入，并返回css代码

less-loader 处理less

sass-loader 处理sass

postcss-loader 用postcss来处理CSS

file-loader 分发文件到output目录并返回相对路径

url-loader 和file-loader类似，但是当文件小于设定的limit时可以返回一个Data Url

**10.什么是闭包，闭包有什么作用？**

希望给一个函数保护一个可反复使用的专属变量，防止被外界修改时，就形成了闭包

原理：外层函数被调用后。外层的作用域链被内层牵引着无法释放就形成了闭包

缺点 : 闭包使函数内部的变量不能被内存释放，这些变量就会占用内存，内存消耗大，可能会 导致内存泄露

**11.从输入URL到浏览器显示页面发生了什么**

<https://www.cnblogs.com/yuanzhiguo/p/8119470.html>

**12.v-show和v-if有什么区别**

(1)相同点：

v-show 和 v-if 都能控制元素的显示和隐藏。

(2)不同点：

a.实现本质方法不同

v-show 本质就是通过设置 css 中的 display 设置为 none，控制隐藏

v-if 是动态的向 DOM 树内添加或者删除 DOM 元素

b.编译的区别

v-show 其实就是在控制 css 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

c.编译的条件

v-show 都会编译，初始值为 false，只是将 display 设为 none，但它也编译了

v-if 初始值为 false，就不会编译了

d.性能比较

v-show 只编译一次，后面其实就是控制 css，而 v-if 不停的销毁和创建，故 v-show性能更好一。

(3)总结（适用场景）：

**如果要频繁切换某节点时，使用 v-show**（无论 true 或者 false 初始都会进行渲染，此后通过 css 来控制显示隐藏，因此切换开销比较小，初始开销较大），**如果不需要频繁切换某节点时，使用 v-if**（因为懒加载，初始为 false 时，不会渲染，但是因为它是通过添加和删除 dom元素来控制显示和隐藏的，因此初始渲染开销较小，切换开销比较大）

**13. 对于mvvm的理解**

MVVM 是 Model-View-ViewModel 的缩写

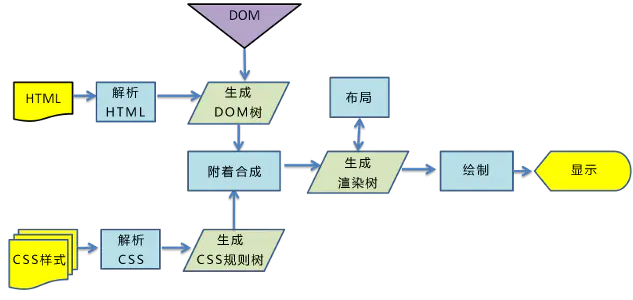
Model:表数据模型,因为它仅仅关注数据本身，不关心任何行为

View:用户操作界面,当ViewModel对Model进行更新的时候，会通过数据绑定更新到View

ViewModel :业务逻辑层,也是View和Model层的桥梁

总结： MVVM模式简化了界面与业务的依赖，解决了数据频繁更新。MVVM 在使用当中，利用双向绑定技术，使得 Model 变化时，ViewModel 会自动更新，而 ViewModel 变化时，View 也会自动变化。

**14. 浏览器解析渲染页面过程**



解析HTML，生成DOM树

解析CSS，生成CSSOM树

将DOM树和CSSOM树关联，生成渲染树(Render Tree)

布局render树（Layout/reflow），负责各元素尺寸、位置的计算

绘制render树（paint），绘制页面像素信息

将像素发送给GPU，展示在页面上。

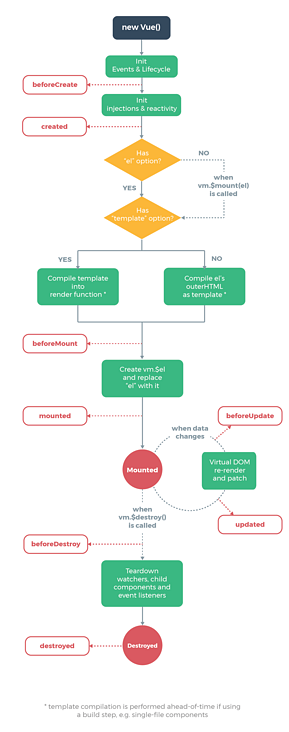
**15. method/computed/watch的使用和区别**

computed：默认computed也是一个watcher具备缓存，只有当依赖的数据变化时才会计算, 当数据没有变化时, 它会读取缓存数据。如果一个数据依赖于其他数据，使用 computed

watch：每次都需要执行函数。 watch 更适用于数据变化时的异步操作。如果需要在某个数据变化时做一些事情，使用watch。

method：只要把方法用到模板上了,每次一变化就会重新渲染视图，性能开销大

**16.Vue的生命周期**



总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： 在beforeCreated阶段，vue实例的挂载元素$el和数据对象data都为undefined，还未初始化。在created阶段，vue实例的数据对象data有了，$el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

**17.如何解决SPA（单页应用）首屏加载速度慢**

首屏时间（First Contentful Paint），指的是浏览器从响应用户输入网址地址，到首屏内容渲染完成的时间，此时整个网页不一定要全部渲染完成，但需要展示当前视窗需要的内容

1）加载慢的原因：网络延迟；资源文件体积过大；资源重复请求加载；加载脚本过程中，渲染堵塞。

2）解决办法：

a、减少入口文件体积：常用的手段是路由懒加载，把不同的路由对应的组件分割成不同的代码块，待路由被请求的时候单独打包路由，使入口文件变小。

b、静态资源本地缓存：后端返回资源问题：采用HTTP缓存，设置Cache-Control，Last-Modified，Etag等响应头。采用Service Worker离线缓存。前端合理利用localStorage。

c、UI框架按需加载

d、避免组件重复打包：在webpack的config文件中，修改CommonsChunkPlugin的配置，设置minChunks为2表示会把使用3次及以上的包抽离出来，放进公共依赖文件，避免了重复加载组件

e、压缩图片资源

f、使用SSR（服务端渲染），即组件和页面通过服务器生成html字符串，再发送到浏览器。vue可以使用Nuxt.js实现服务端渲染。