1. Js
2. **数据类型主要包括两部分：**

基本数据类型： Undefined、Null、Boolean、Number 和 String

引用数据类型： Object (包括 Object 、Array 、Function)

ECMAScript 2015 新增:Symbol(创建后独一无二且不可变的数据类型 )

1. **判断一个值是什么类型有哪些方法？**

typeof 运算符

instanceof 运算符

Object.prototype.toString 方法

1. **null 和 undefined 的区别？**

null 表示一个对象被定义了，值为“空值”；  
undefined 表示不存在这个值。  
（1）变量被声明了，但没有赋值时，就等于undefined。 （2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。 （3）对象没有赋值的属性，该属性的值为undefined。 （4）函数没有返回值时，默认返回undefined。

1. **怎么判断一个变量arr的话是否为数组（此题用 typeof 不行）？**

arr instanceof Array  
arr.constructor == Array  
Object.protype.toString.call(arr) == '[Object Array]'

1. **“ ===”、“ ==”的区别？**

==，当且仅当两个运算数相等时，它返回 true，即不检查数据类型  
===，只有在无需类型转换运算数就相等的情况下，才返回 true，需要检查数据类型

1. **“eval是做什么的？**

它的功能是把对应的字符串解析成 JS 代码并运行；  
应该避免使用 eval，不安全，非常耗性能（2次，一次解析成 js 语句，一次执行）。

1. **箭头函数有哪些特点？**

不需要function关键字来创建函数  
省略return关键字  
改变this指向

1. **var、let、const 区别？**

var 存在变量提升。  
let 只能在块级作用域内访问。  
const 用来定义常量，必须初始化，不能修改（对象特殊）

1. **new操作符具体干了什么呢？**

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。  
2、属性和方法被加入到 this 引用的对象中。  
3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

1. **documen.write 和 innerHTML 的区别？**

document.write 只能重绘整个页面  
innerHTML 可以重绘页面的一部分

1. **ajax过程？**

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.  
(2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.  
(3)设置响应HTTP请求状态变化的函数.  
(4)发送HTTP请求.  
(5)获取异步调用返回的数据.  
(6)使用JavaScript和DOM实现局部刷新.

**12. 介绍一下闭包和闭包常用场景？**

闭包指的是一个函数可以访问另一个函数作用域中变量。常见的构造方法，是在一个函数内部定义另外一个函数。内部函数可以引用外层的变量；外层变量不会被垃圾回收机制回收。 注意，闭包的原理是作用域链，所以闭包访问的上级作用域中的变量是个对象，其值为其运算结束后的最后一个值。 优点：避免全局变量污染。缺点：容易造成内存泄漏。

闭包是指有权访问另一个函数作用域中的变量的函数，创建闭包常见方式，就是在一个函数的内部创建另一个函数

使用闭包主要为了设计私有的方法和变量，闭包的优点是可以避免变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念。

闭包有三个特性：

函数嵌套函数

函数内部可以引用外部的参数和变量

参数和变量不会被垃圾回收机制回收

应用场景，设置私有变量的方法

不适用场景：返回闭包的函数是个非常大的函数

闭包的缺点就是常驻内存，会增大内存使用量，使用不当会造成内存泄漏

**13. JavaScript原型，原型链 ? 有什么特点？**

每个对象都会在其内部初始化一个属性，就是prototype(原型)，当我们访问一个对象的属性时，  
如果这个对象内部不存在这个属性，那么他就会去prototype里找这个属性，这个prototype又会有自己的prototype，  
于是就这样一直找下去，也就是我们平时所说的原型链的概念。

关系：instance.constructor.prototype = instance.proto

特点：  
JavaScript对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

1. **防抖节流**

防抖（debounce）

所谓防抖，就是指触发事件后在 n 秒内函数只能执行一次，如果在 n 秒内又触发了事件，则会重新计算函数执行时间。

防抖函数分为非立即执行版和立即执行版。

节流（throttle）

所谓节流，就是指连续触发事件但是在 n 秒中只执行一次函数。节流会稀释函数的执行频率。

对于节流，一般有两种方式可以实现，分别是时间戳版和定时器版。

1. **深浅拷贝**

浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存。

1.Object.assign()

2.Array.prototype.concat()

3.Array.prototype.slice()

深拷贝会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。

1.JSON.parse(JSON.stringify())

2.手写递归方法 3.函数库lodash

**16.冒泡**

addEventListener()

Settimeout()

1. **判断数组**

Instanceof

Constructor

Object.prototype.toString.call()

Array.isArray()

1. **数组去重**

Set

indexOf

1. **JS垃圾回收机制？**

1) 标记清除： 这个算法把“对象是否不再需要”简化定义为“对象是否可以获得”。 这个算法假定设置一个叫做根（root）的对象（在Javascript里，根是全局对象）。定期的，垃圾回收器将从根开始，找所有从根开始引用的对象，然后找这些对象引用的对象。从根开始，垃圾回收器将找到所有可以获得的对象和所有不能获得的对象。

2) 引用计数： 这是最简单的垃圾收集算法。此算法把“对象是否不再需要”简化定义为“对象有没有其他对象引用到它”。如果没有引用指向该对象（零引用），对象将被垃圾回收机制回收。 该算法有个限制：无法处理循环引用。两个对象被创建，并互相引用，形成了一个循环。它们被调用之后不会离开函数作用域，所以它们已经没有用了，可以被回收了。然而，引用计数算法考虑到它们互相都有至少一次引用，所以它们不会被回收。

**19.回流与重绘**

当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建。这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树。完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程成为重绘

20. call和apply call（）方法和apply（）方法的作用相同，动态改变某个类的某个方法的运行环境。他们的区别在于接收参数的方式不同。在使用call（）方法时，传递给函数的参数必须逐个列举出来。使用apply（）时，传递给函数的是参数数组。

1. **1说一下你对同步和异步的理解**

答：同步即sync，形象的说就是代码一行行执行，前面代码和请求没有执行完，后面的代码和请求就不会被执行，

缺点：容易导致代码阻塞

优点：程序员容易理解（因为代码从上往下一行行执行，强调顺序）

异步：即async,形象的说就是代码可以在当前程序没有执行完，也可以执行后面的代码

缺点：程序员不易理解（因为不是按顺序执行的）

优点：可以解决代码阻塞问题，提升代码执行效率和性能

异步解决方案主要有三个：

1. 回调函数
2. promise（重点掌握）
3. generator(了解)
4. async和await（重点掌握）

**1 介绍[JavaScript](https://cloud.tencent.com/product/sms?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)的基本数据类型** Number、String 、Boolean 、Null、Undefined Object 是 JavaScript 中所有对象的父对象 数据封装类对象：Object、Array、Boolean、Number 和 String 其他对象：Function、Arguments、Math、Date、RegExp、Error 新类型：Symbol

**2 说说写JavaScript的基本规范？** 1) 不要在同一行声明多个变量 2) 使用 ===或!==来比较true/false或者数值 3) switch必须带有default分支 4) 函数应该有返回值 5) for if else 必须使用大括号 6) 语句结束加分号 7) 命名要有意义，使用驼峰命名法

**3 jQuery使用建议** 1) 尽量减少对dom元素的访问和操作 2) 尽量避免给dom元素绑定多个相同类型的事件处理函数，可以将多个相同类型事件 处理函数合并到一个处理函数，通过数据状态来处理分支 3) 尽量避免使用toggle事件

**4 Ajax使用** 全称 ： Asynchronous Javascript And XML 所谓异步，就是向服务器发送请求的时候，我们不必等待结果，而是可以同时做其他的事情，等到有了结果它自己会根据设定进行后续操作，与此同时，页面是不会发生整页刷新的，提高了用户体验。 创建Ajax的过程： 1) 创建XMLHttpRequest对象（异步调用对象）

var xhr = new XMLHttpRequest();

2) 创建新的Http请求（方法、URL、是否异步）

xhr.open(‘get’,’example.php’,false);

3) 设置响应HTTP请求状态变化的函数。 onreadystatechange事件中readyState属性等于4。响应的HTTP状态为200(OK)或者304(Not Modified)。 4) 发送http请求

xhr.send(data);

5) 获取异步调用返回的数据 注意： 1) 页面初次加载时，尽量在web服务器一次性输出所有相关的数据，只在页面加载完成之后，用户进行操作时采用ajax进行交互。 2) 同步ajax在IE上会产生页面假死的问题。所以建议采用异步ajax。 3) 尽量减少ajax请求次数 4) ajax安全问题，对于敏感数据在服务器端处理，避免在客户端处理过滤。对于关键业务逻辑代码也必须放在服务器端处理。

**5 JavaScript有几种类型的值？**你能画一下他们的内存图吗？ 基本数据类型存储在栈中，引用数据类型（对象）存储在堆中，指针放在栈中。 两种类型的区别是：存储位置不同；原始数据类型直接存储在栈中的简单数据段，占据空间小、大小固定，属于被频繁使用数据，所以放入栈中存储；引用数据类型存储在堆中的对象,占据空间大、大小不固定,如果存储在栈中，将会影响程序运行的性能 引用数据类型在栈中存储了指针，该指针指向堆中该实体的起始地址。当解释器寻找引用值时，会首先检索其在栈中的地址，取得地址后从堆中获得实体。

**6 栈和堆的区别？** 栈（stack）：由编译器自动分配释放，存放函数的参数值，局部变量等； 堆（heap）：一般由程序员分配释放，若程序员不释放，程序结束时可能由操作系统释放。

**7 Javascript实现继承的几种方式** 可以参考我的另一篇文章[JavaScript实现类与继承的方法（全面整理）](https://segmentfault.com/a/1190000013253890" \t "https://cloud.tencent.com/developer/article/_blank)

**8 Javascript创建对象的几种方式？** 可以参考我的另一篇文章[JavaScript实现类与继承的方法（全面整理）](https://segmentfault.com/a/1190000013253890" \t "https://cloud.tencent.com/developer/article/_blank)

**9 Javascript作用链域** 作用域链的原理和原型链很类似，如果这个变量在自己的作用域中没有，那么它会寻找父级的，直到最顶层。 注意：JS没有块级作用域，若要形成块级作用域，可通过（function（）｛｝）（）；立即执行的形式实现。

**10 谈谈this的理解** 1) this总是指向函数的直接调用者（而非间接调用者） 2) 如果有new关键字，this指向new出来的那个对象 3) 在事件中，this指向目标元素，特殊的是IE的attachEvent中的this总是指向全局对象window。

**11 eval是做什么的？** 它的功能是把对应的字符串解析成JS代码并运行；应该避免使用eval，不安全，非常耗性能（2次，一次解析成js语句，一次执行）。

**12 什么是window对象?** 什么是document对象? window对象代表浏览器中打开的一个窗口。document对象代表整个html文档。实际上，document对象是window对象的一个属性。

**13 null，undefined的区别？** null表示一个对象被定义了，但存放了空指针，转换为数值时为0。 undefined表示声明的变量未初始化，转换为数值时为NAN。 typeof(null) -- object; typeof(undefined) -- undefined

**14 写一个通用的事件侦听器函数(机试题)**

**15 ["1", "2", "3"].map(parseInt) 答案是多少？** [1,NaN,NaN]

解析： Array.prototype.map() array.map(callback[, thisArg]) callback函数的执行规则 参数：自动传入三个参数 currentValue（当前被传递的元素）； index（当前被传递的元素的索引）； array（调用map方法的数组）

parseInt方法接收两个参数 第三个参数["1", "2", "3"]将被忽略。parseInt方法将会通过以下方式被调用 parseInt("1", 0) parseInt("2", 1) parseInt("3", 2)

parseInt的第二个参数radix为0时，ECMAScript5将string作为十进制数字的字符串解析； parseInt的第二个参数radix为1时，解析结果为NaN； parseInt的第二个参数radix在2—36之间时，如果string参数的第一个字符（除空白以外），不属于radix指定进制下的字符，解析结果为NaN。 parseInt("3", 2)执行时，由于"3"不属于二进制字符，解析结果为NaN。

**16 关于事件，IE与火狐的事件机制有什么区别？ 如何阻止冒泡？** IE为事件冒泡，Firefox同时支持事件捕获和事件冒泡。但并非所有浏览器都支持事件捕获。jQuery中使用event.stopPropagation()方法可阻止冒泡;（旧IE的方法 ev.cancelBubble = true;）

**17 什么是闭包（closure），为什么要用它？** 闭包指的是一个函数可以访问另一个函数作用域中变量。常见的构造方法，是在一个函数内部定义另外一个函数。内部函数可以引用外层的变量；外层变量不会被垃圾回收机制回收。 注意，闭包的原理是作用域链，所以闭包访问的上级作用域中的变量是个对象，其值为其运算结束后的最后一个值。 优点：避免全局变量污染。缺点：容易造成内存泄漏。 例子：

function makeFunc() {

var name = "Mozilla";

function displayName() {

console.log(name);

}

return displayName;}var myFunc = makeFunc();myFunc(); //输出Mozilla

myFunc 变成一个 闭包。闭包是一种特殊的对象。它由两部分构成：函数，以及创建该函数的环境。环境由闭包创建时在作用域中的任何局部变量组成。在我们的例子中，myFunc 是一个闭包，由 displayName 函数和闭包创建时存在的 "Mozilla" 字符串形成。

**18 javascript 代码中的"use strict";是什么意思 ?** 使用它区别是什么？ 除了正常模式运行外，ECMAscript添加了第二种运行模式：“严格模式”。 作用： 1) 消除js不合理，不严谨地方，减少怪异行为 2) 消除代码运行的不安全之处， 3) 提高编译器的效率，增加运行速度 4) 为未来的js新版本做铺垫。

**19 如何判断一个对象是否属于某个类？** 使用instanceof 即if(a instanceof Person){alert('yes');}

**20 new操作符具体干了什么呢?** 1) 创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。 2) 属性和方法被加入到 this 引用的对象中。 3) 新创建的对象由 this 所引用，并且最后隐式的返回 this 。

**21 Javascript中，执行时对象查找时，永远不会去查找原型的函数？** Object.hasOwnProperty(proName)：是用来判断一个对象是否有你给出名称的属性。不过需要注意的是，此方法无法检查该对象的原型链中是否具有该属性，该属性必须是对象本身的一个成员。

**22 对JSON的了解？** 全称：JavaScript Object Notation JSON中对象通过“{}”来标识，一个“{}”代表一个对象，如{“AreaId”:”123”}，对象的值是键值对的形式（key：value）。JSON是JS的一个严格的子集，一种轻量级的数据交换格式，类似于xml。数据格式简单，易于读写，占用带宽小。 两个函数： JSON.parse(str) 解析JSON字符串 把JSON字符串变成JavaScript值或对象 JSON.stringify(obj) 将一个JavaScript值(对象或者数组)转换为一个 JSON字符串 eval('('＋json＋')') 用eval方法注意加括号 而且这种方式更容易被攻击

**23 JS延迟加载的方式有哪些？** JS的延迟加载有助与提高页面的加载速度。 defer和async、动态创建DOM方式（用得最多）、按需异步载入JS defer：延迟脚本。立即下载，但延迟执行（延迟到整个页面都解析完毕后再运行），按照脚本出现的先后顺序执行。 async：异步脚本。下载完立即执行，但不保证按照脚本出现的先后顺序执行。

**24 同步和异步的区别?** 同步的概念在操作系统中：不同进程协同完成某项工作而先后次序调整（通过阻塞、唤醒等方式），同步强调的是顺序性，谁先谁后。异步不存在顺序性。 同步：浏览器访问服务器，用户看到页面刷新，重新发请求，等请求完，页面刷新，新内容出现，用户看到新内容之后进行下一步操作。 异步：浏览器访问服务器请求，用户正常操作，浏览器在后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容。

25 什么是跨域问题 ，如何解决跨域问题? 可以参考我的另一篇文章[什么是跨域以及几种简单解决方案](https://segmentfault.com/a/1190000013278814" \t "https://cloud.tencent.com/developer/article/_blank)

**26 页面编码和被请求的资源编码如果不一致如何处理？** 若请求的资源编码，如外引js文件编码与页面编码不同。可根据外引资源编码方式定义为 charset="utf-8"或"gbk"。 比如：[http://www.yyy.com/a.html](http://www.yyy.com/a.html" \t "https://cloud.tencent.com/developer/article/_blank) 中嵌入了一个[http://www.xxx.com/test.js](http://www.xxx.com/test.js" \t "https://cloud.tencent.com/developer/article/_blank) a.html 的编码是gbk或gb2312的。 而引入的js编码为utf-8的 ，那就需要在引入的时候 <script src="[http://www.xxx.com/test.js&quot](http://www.xxx.com/test.js&quot" \t "https://cloud.tencent.com/developer/article/_blank); charset="utf-8"></script>

**27 模块化开发怎么做？** 模块化开发指的是在解决某一个复杂问题或者一系列问题时，依照一种分类的思维把问题进行系统性的分解。模块化是一种将复杂系统分解为代码结构更合理，可维护性更高的可管理的模块方式。对于软件行业：系统被分解为一组高内聚，低耦合的模块。 （1）定义封装的模块 （2）定义新模块对其他模块的依赖 （3）可对其他模块的引入支持。在JavaScript中出现了一些非传统模块开发方式的规范。 CommonJS的模块规范，AMD（Asynchronous Module Definition），CMD（Common Module Definition）等。AMD是异步模块定义，所有的模块将被异步加载，模块加载不影响后边语句运行。

**28 AMD（Modules/Asynchronous-Definition）、CMD（Common Module Definition）规范区别？** AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。CMD 是 SeaJS 在推广过程中对模块定义的规范化产出。 区别： 1) 对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。不过 RequireJS 从 2.0 开始，也改成可以延迟执行（根据写法不同，处理方式不同）。 2) CMD 推崇依赖就近，AMD 推崇依赖前置。 3) AMD 的 API 默认是一个当多个用，CMD 的 API 严格区分，推崇职责单一。

// CMDdefine(function(require, exports, module) {

var a = require('./a')

a.doSomething()

// 此处略去 100 行

var b = require('./b') // 依赖可以就近书写

b.doSomething()})// AMD 默认推荐define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好

a.doSomething();

// 此处略去 100 行

b.doSomething();})

核心是js的加载模块，通过正则匹配模块以及模块的依赖关系，保证文件加载的先后顺序，根据文件的路径对加载过的文件做了缓存。

在使用call（）方法时，传递给函数的参数必须逐个列举出来。使用apply（）时，传递给函数的是参数数组。

**29 requireJS的核心原理是什么？（如何动态加载的？如何避免多次加载的？如何缓存的？）**

**30 call和apply call（）方法和apply（）方法的作用相同，动态改变某个类的某个方法的运行环境。他们的区别在于接收参数的方式不同。**

**31 谈一谈你对ECMAScript6的了解**

**32 documen.write和 innerHTML的区别 document.write()只能重绘整个页面**

**setTimeout(function(){**

**document.write('<p>5 secs later</p>');}, 5000);**

**或**

**window.onload = function() { document.write("HI");**

**innerHTML可以重绘页面的一部分**

**33 回流与重绘**

当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建。这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树。完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程成为重绘

**34 DOM操作** （1）创建新节点 createDocumentFragment() //创建一个DOM片段 createElement() //创建一个具体的元素 createTextNode() //创建一个文本节点 （2）添加、移除、替换、插入 appendChild() removeChild() replaceChild() insertBefore() //在已有的子节点前插入一个新的子节点 （3）查找 getElementsByTagName() //通过标签名称 getElementsByName() //通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包括id等于name值的) getElementById() //通过元素Id，唯一性

**35 数组对象有哪些原生方法，**列举一下 pop、push、shift、unshift、splice、reverse、sort、concat、join、slice、toString、indexOf、lastIndexOf、reduce、reduceRight forEach、map、filter、every、some

**36 那些操作会造成内存泄漏** 全局变量、闭包、DOM清空或删除时，事件未清除、子元素存在引用

**37 什么是Cookie 隔离？**（或者：请求资源的时候不要带cookie怎么做） 通过使用多个非主要域名来请求静态文件，如果静态文件都放在主域名下，那静态文件请求的时候带有的cookie的数据提交给server是非常浪费的，还不如隔离开。因为cookie有域的限制，因此不能跨域提交请求，故使用非主要域名的时候，请求头中就不会带有cookie数据，这样可以降低请求头的大小，降低请求时间，从而达到降低整体请求延时的目的。同时这种方式不会将cookie传入server，也减少了server对cookie的处理分析环节，提高了server的http请求的解析速度。

**38 响应事件** onclick鼠标点击某个对象；onfocus获取焦点；onblur失去焦点；onmousedown鼠标被按下

**39 flash和js通过什么类如何交互?** Flash提供了ExternalInterface接口与JavaScript通信，ExternalInterface有两个方法，call和addCallback，call的作用是让Flash调用js里的方法，addCallback是用来注册flash函数让js调用。

**40 Flash与Ajax各自的优缺点？** Flash：适合处理多媒体、矢量图形、访问机器。但对css、处理文本不足，不容易被搜索。 Ajax：对css、文本支持很好，但对多媒体、矢量图形、访问机器不足。

**41 有效的javascript变量定义规则** 第一个字符必须是一个字母、下划线（\_）或一个美元符号（$）；其他字符可以是字母、下划线、美元符号或数字。

**42 XML与JSON的区别？** 1) 数据体积方面。JSON相对于XML来讲，数据的体积小，传递的速度更快些。 2) 数据交互方面。JSON与JavaScript的交互更加方便，更容易解析处理，更好的数据交互。 3) 数据描述方面。JSON对数据的描述性比XML较差。 4) 传输速度方面。JSON的速度要远远快于XML。

**43 HTML与XML的区别？** （1）XML用来传输和存储数据，HTML用来显示数据； （2）XML使用的标签不用预先定义 （3）XML标签必须成对出现 （4）XML对大小写敏感 （5）XML中空格不会被删减 （6）XML中所有特殊符号必须用编码表示 （7）XML中的图片必须有文字说明

**44 渐进增强与优雅降级 渐进增强**：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进，达到更好的用户体验。 优雅降级：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

**45 Web Worker和Web Socket？** web socket：在一个单独的持久连接上提供全双工、双向的通信。使用自定义的协议（ws://、wss://），同源策略对web socket不适用。 web worker：运行在后台的JavaScript，不影响页面的性能。 创建worker：var worker = new Worker(url); 向worker发送数据：worker.postMessage(data); 接收worker返回的数据：worker.onmessage 终止一个worker的执行：worker.terminate();

**46 JS垃圾回收机制？** 1) 标记清除： 这个算法把“对象是否不再需要”简化定义为“对象是否可以获得”。 这个算法假定设置一个叫做根（root）的对象（在Javascript里，根是全局对象）。定期的，垃圾回收器将从根开始，找所有从根开始引用的对象，然后找这些对象引用的对象。从根开始，垃圾回收器将找到所有可以获得的对象和所有不能获得的对象。

2) 引用计数： 这是最简单的垃圾收集算法。此算法把“对象是否不再需要”简化定义为“对象有没有其他对象引用到它”。如果没有引用指向该对象（零引用），对象将被垃圾回收机制回收。 该算法有个限制：无法处理循环引用。两个对象被创建，并互相引用，形成了一个循环。它们被调用之后不会离开函数作用域，所以它们已经没有用了，可以被回收了。然而，引用计数算法考虑到它们互相都有至少一次引用，所以它们不会被回收。

**47 web应用从服务器主动推送data到客户端的方式？** JavaScript数据推送：commet（基于http长连接的服务器推送技术）。 基于web socket的推送：SSE（server-send Event）

**48 如何删除一个cookie？** 1） 将cookie的失效时间设置为过去的时间（expires）

document.cookie = ‘user=’+ encodeURIComponent(‘name’) + ';

expires=’+ new Date(0);

2） 将系统时间设置为当前时间往前一点时间

var data = new Date();

date.setDate(date.getDate()-1);

**49 attribute与property的区别？** attribute是dom元素在文档中作为html标签拥有的属性 property是dom元素在js中作为对象拥有的属性。 所以，对于html的标准属性来说，attribute和property是同步的，是会自动更新的。但对于自定义属性，他们不同步。

**50 Ajax请求的页面历史记录状态问题？** （1）通过location.hash记录状态，让浏览器记录Ajax请求时页面状态的变化。 （2）通过HTML5的history.pushstate，来实现浏览器地址栏的无刷新改变。

**从浏览器地址栏输入 url 到显示页面的步骤**

基础版本：

浏览器根据请求的 URL 交给 DNS 域名解析，找到真实 IP ，向服务器发起请求；

服务器交给后台处理完成后返回数据，浏览器接收⽂件（ HTML、JS、CSS 、图象等）；

浏览器对加载到的资源（ HTML、JS、CSS 等）进⾏语法解析，建立相应的内部数据结构 （如 HTML 的 DOM）；

载⼊解析到的资源⽂件，渲染页面，完成。

详细版：

1、在浏览器地址栏输⼊URL

2、浏览器查看缓存，如果请求资源在缓存中并且新鲜，跳转到转码步骤

如果资源未缓存，发起新请求

如果已缓存，检验是否⾜够新鲜，⾜够新鲜直接提供给客户端，否则与服务器进⾏验证。

检验新鲜通常有两个HTTP头进⾏控制 Expires 和 Cache-Control：

HTTP1.0提供 Expires，值为⼀个绝对时间表示缓存新鲜⽇期

HTTP1.1增加了Cache-Control: max-age=time，值为以秒为单位的最⼤新鲜时间

3、浏览器解析URL获取协议，主机，端⼝，path

4、浏览器组装⼀个HTTP（GET）请求报⽂

5、浏览器获取主机 ip 地址，过程如下：

浏览器缓存

本机缓存

hosts⽂件

路由器缓存

ISP DNS缓存

DNS递归查询（可能存在负载均衡导致每次IP不⼀样）

6、打开⼀个socket与⽬标IP地址，端⼝建⽴TCP链接，三次握⼿如下：

客户端发送⼀个TCP的SYN=1，Seq=X的包到服务器端口

服务器发回SYN=1， ACK=X+1， Seq=Y的响应包

客户端发送ACK=Y+1， Seq=Z

7、TCP链接建⽴后发送HTTP请求

8、服务器接受请求并解析，将请求转发到服务程序，如虚拟主机使⽤HTTP Host头部判断请求的服务程序

9、服务器检查\*\*HTTP请求头是否包含缓存验证信息\*\*，如果验证缓存新鲜，返回304等对应状态码

10、处理程序读取完整请求并准备HTTP响应，可能需要查询数据库等操作

11、服务器将响应报⽂通过TCP连接发送回浏览器

12、浏览器接收HTTP响应，然后根据情况选择关闭TCP连接或者保留重⽤，关闭TCP连接的四次握⼿如下：

主动⽅发送Fin=1， Ack=Z， Seq= X报⽂

被动⽅发送ACK=X+1， Seq=Z报⽂

被动⽅发送Fin=1， ACK=X， Seq=Y报⽂

主动⽅发送ACK=Y， Seq=X报⽂

13、浏览器检查响应状态吗：是否为1XX，3XX， 4XX， 5XX，这些情况处理与2XX不同

14、如果资源可缓存，进行缓存

15、对响应进行解码（例如gzip压缩）

16、根据资源类型决定如何处理（假设资源为HTML⽂档）

17、解析HTML⽂档，构件DOM树，下载资源，构造CSSOM树，执⾏js脚本，这些操作没有严 格的先后顺序，以下分别解释：

16、构建DOM树：

Tokenizing：根据HTML规范将字符流解析为标记

Lexing：词法分析将标记转换为对象并定义属性和规则

DOM construction：根据HTML标记关系将对象组成DOM树

17、解析过程中遇到图⽚、样式表、js⽂件，启动下载

18、构建CSSOM树：

Tokenizing：字符流转换为标记流

Node：根据标记创建节点

CSSOM：节点创建CSSOM树

19、根据DOM树和CSSOM树构建渲染树 :

从DOM树的根节点遍历所有可⻅节点，不可⻅节点包括：

script , meta 这样本身 不可⻅的标签。

被css隐藏的节点，如 display: none

对每⼀个可⻅节点，找到恰当的CSSOM规则并应⽤

发布可视节点的内容和计算样式

20、js解析如下：

浏览器创建Document对象并解析HTML，将解析到的元素和⽂本节点添加到⽂档中，此时\*\*document.readystate为loading\*\*

HTML解析器遇到没有async和defer的script时，将他们添加到⽂档中，然后执⾏⾏内 或外部脚本。这些脚本会同步执⾏，并且在脚本下载和执⾏时解析器会暂停。这样就可以⽤document.write()把⽂本插⼊到输⼊流中。同步脚本经常简单定义函数和注册事件处理程序，他们可以遍历和操作script和他们之前的⽂档内容

当解析器遇到设置了async属性的script时，开始下载脚本并继续解析⽂档。脚本会在它 下载完成后尽快执⾏，但是解析器不会停下来等它下载。异步脚本禁止使⽤ document.write()，它们可以访问⾃⼰script和之前的⽂档元素

当⽂档完成解析，document.readState变成interactive

所有defer脚本会按照在⽂档出现的顺序执⾏，延迟脚本能访问完整⽂档树，禁止使⽤ document.write()

浏览器在Document对象上触发DOMContentLoaded事件

此时⽂档完全解析完成，浏览器可能还在等待如图⽚等内容加载，等这些内容完成载⼊ 并且所有异步脚本完成载⼊和执⾏，document.readState变为complete，window触发 load事件

21、显示⻚⾯（HTML解析过程中会逐步显示⻚⾯）

详细简版：

从浏览器接收 url 到开启⽹络请求线程（这⼀部分可以展开浏览器的机制以及进程与线程 之间的关系）

开启⽹络线程到发出⼀个完整的 HTTP 请求（这⼀部分涉及到dns查询， TCP/IP 请求，五层因特⽹协议栈等知识）

从服务器接收到请求到对应后台接收到请求（这⼀部分可能涉及到负载均衡，安全拦截以及后台内部的处理等等）

后台和前台的 HTTP 交互（这⼀部分包括 HTTP 头部、响应码、报⽂结构、 cookie 等知 识，可以提下静态资源的 cookie 优化，以及编码解码，如 gzip 压缩等）

单独拎出来的缓存问题， HTTP 的缓存（这部分包括http缓存头部， ETag ， catchcontrol 等）

浏览器接收到 HTTP 数据包后的解析流程（解析 html、 词法分析然后解析成 dom 树、解析 css ⽣成 css 规则树、合并成 render 树，然后 layout 、 painting 渲染、复合图层的合成、 GPU 绘制、外链资源的处理、 loaded 和 DOMContentLoaded 等）

CSS 的可视化格式模型（元素的渲染规则，如包含块，控制框， BFC ， IFC 等概念）

JS 引擎解析过程（ JS 的解释阶段，预处理阶段，执⾏阶段⽣成执⾏上下⽂， VO ，作 ⽤域链、回收机制等等）

其它（可以拓展不同的知识模块，如跨域，web安全， hybrid 模式等等内容）

**一、cookie和session**

cookie和session都是用来跟踪浏览器用户身份的会话方式。

区别：

1、保持状态：cookie保存在浏览器端，session保存在服务器端

2、使用方式：

（1）cookie机制：如果不在浏览器中设置过期时间，cookie被保存在内存中，生命周期随浏览器的关闭而结束，这种cookie简称会话cookie。如果在浏览器中设置了cookie的过期时间，cookie被保存在硬盘中，关闭浏览器后，cookie数据仍然存在，直到过期时间结束才消失。

     Cookie是服务器发给客户端的特殊信息，cookie是以文本的方式保存在客户端，每次请求时都带上它

（2）session机制：当服务器收到请求需要创建session对象时，首先会检查客户端请求中是否包含sessionid。如果有sessionid，服务器将根据该id返回对应session对象。如果客户端请求中没有sessionid，服务器会创建新的session对象，并把sessionid在本次响应中返回给客户端。通常使用cookie方式存储sessionid到客户端，在交互中浏览器按照规则将sessionid发送给服务器。如果用户禁用cookie，则要使用URL重写，可以通过response.encodeURL(url) 进行实现；API对encodeURL的结束为，当浏览器支持Cookie时，url不做任何处理；当浏览器不支持Cookie的时候，将会重写URL将SessionID拼接到访问地址后。

3、存储内容：cookie只能保存字符串类型，以文本的方式；session通过类似与Hashtable的数据结构来保存，能支持任何类型的对象(session中可含有多个对象)

4、存储的大小：cookie：单个cookie保存的数据不能超过4kb；session大小没有限制。

5、安全性：cookie：针对cookie所存在的攻击：Cookie欺骗，Cookie截获；session的安全性大于cookie。

　　　　　　原因如下：（1）sessionID存储在cookie中，若要攻破session首先要攻破cookie；

　　　　　　　　　　　（2）sessionID是要有人登录，或者启动session\_start才会有，所以攻破cookie也不一定能得到sessionID；

　　　　　　　　　　　（3）第二次启动session\_start后，前一次的sessionID就是失效了，session过期后，sessionID也随之失效。

　　　　　　　　　　　（4）sessionID是加密的

　　　　　　　　　　　（5）综上所述，攻击者必须在短时间内攻破加密的sessionID，这很难。

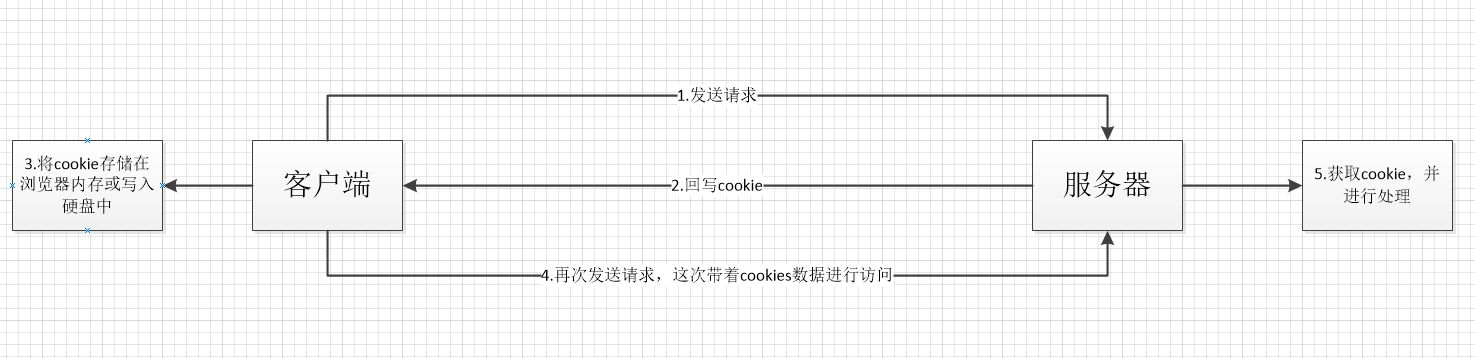
6、应用场景：

cookie：（1）判断用户是否登陆过网站，以便下次登录时能够实现自动登录（或者记住密码）。如果我们删除cookie，则每次登录必须从新填写登录的相关信息。

　　　　（2）保存上次登录的时间等信息。

　　　　（3）保存上次查看的页面

　　　　（4）浏览计数



session：Session用于保存每个用户的专用信息，变量的值保存在服务器端，通过SessionID来区分不同的客户。

　　（1）网上商城中的购物车

　　（2）保存用户登录信息

　　（3）将某些数据放入session中，供同一用户的不同页面使用

　　（4）防止用户非法登录

 7、缺点：cookie：（1）大小受限

　　　　　　　　（2）用户可以操作（禁用）cookie，使功能受限

　　　　　　　　（3）安全性较低

　　　　　　　　（4）有些状态不可能保存在客户端。

　　　　　　　　（5）每次访问都要传送cookie给服务器，浪费带宽。

　　　　　　　　（6）cookie数据有路径（path）的概念，可以限制cookie只属于某个路径下。

 　　　　session：（1）Session保存的东西越多，就越占用服务器内存，对于用户在线人数较多的网站，服务器的内存压力会比较大。

　　　　　　　　（2）依赖于cookie（sessionID保存在cookie），如果禁用cookie，则要使用URL重写，不安全

　　　　　　　　（3）创建Session变量有很大的随意性，可随时调用，不需要开发者做精确地处理，所以，过度使用session变量将会导致代码不可读而且不好维护。

二、WebStorage

WebStorage的目的是克服由cookie所带来的一些限制，当数据需要被严格控制在客户端时，不需要持续的将数据发回服务器。

WebStorage两个主要目标：（1）提供一种在cookie之外存储会话数据的路径。（2）提供一种存储大量可以跨会话存在的数据的机制。

HTML5的WebStorage提供了两种API：localStorage（本地存储）和sessionStorage（会话存储）。

1、生命周期：localStorage:localStorage的生命周期是永久的，关闭页面或浏览器之后localStorage中的数据也不会消失。localStorage除非主动删除数据，否则数据永远不会消失。

　　　　　　  sessionStorage的生命周期是在仅在当前会话下有效。sessionStorage引入了一个“浏览器窗口”的概念，sessionStorage是在同源的窗口中始终存在的数据。只要这个浏览器窗口没有关闭，即使刷新页面或者进入同源另一个页面，数据依然存在。但是sessionStorage在关闭了浏览器窗口后就会被销毁。同时独立的打开同一个窗口同一个页面，sessionStorage也是不一样的。

2、存储大小：localStorage和sessionStorage的存储数据大小一般都是：5MB

3、存储位置：localStorage和sessionStorage都保存在客户端，不与服务器进行交互通信。

4、存储内容类型：localStorage和sessionStorage只能存储字符串类型，对于复杂的对象可以使用ECMAScript提供的JSON对象的stringify和parse来处理

5、获取方式：localStorage：window.localStorage;；sessionStorage：window.sessionStorage;。

6、应用场景：localStoragese：常用于长期登录（+判断用户是否已登录），适合长期保存在本地的数据。sessionStorage：敏感账号一次性登录；

WebStorage的优点：

（1）存储空间更大：cookie为4KB，而WebStorage是5MB；

（2）节省网络流量：WebStorage不会传送到服务器，存储在本地的数据可以直接获取，也不会像cookie一样美词请求都会传送到服务器，所以减少了客户端和服务器端的交互，节省了网络流量；

（3）对于那种只需要在用户浏览一组页面期间保存而关闭浏览器后就可以丢弃的数据，sessionStorage会非常方便；

（4）快速显示：有的数据存储在WebStorage上，再加上浏览器本身的缓存。获取数据时可以从本地获取会比从服务器端获取快得多，所以速度更快；

（5）安全性：WebStorage不会随着HTTP header发送到服务器端，所以安全性相对于cookie来说比较高一些，不会担心截获，但是仍然存在伪造问题；

（6）WebStorage提供了一些方法，数据操作比cookie方便；

　　　　setItem (key, value) ——  保存数据，以键值对的方式储存信息。

      　　 getItem (key) ——  获取数据，将键值传入，即可获取到对应的value值。

        　　removeItem (key) ——  删除单个数据，根据键值移除对应的信息。

        　　clear () ——  删除所有的数据

        　　key (index) —— 获取某个索引的key

1. Vue

**1. 谈谈你对MVVM开发模式的理解？**

MVVM分为Model、View、ViewModel三者。  
Model 代表数据模型，数据和业务逻辑都在Model层中定义；  
View 代表UI视图，负责数据的展示；  
ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理用户交互操作；  
Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。  
这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作 dom。

**2. v-if 和 v-show 有什么区别？**

v-if 是真正的条件渲染，会控制这个 DOM 节点的存在与否。因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建；也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 的 “display” 属性进行切换。

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。

**nextTick？**

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

所以就衍生出了这个获取更新后的DOM的Vue方法。所以放在Vue.nextTick()回调函数中的执行的应该是会对DOM进行操作的 js代码；

理解：nextTick()，是将回调函数延迟在下一次dom更新数据后调用，简单的理解是：当数据更新了，在dom中渲染后，自动执行该函数

**什么时候需要用的Vue.nextTick()？？**

1. Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中，原因是在created()钩子函数执行的时候DOM 其实并未进行任何渲染，而此时进行DOM操作无异于徒劳，所以此处一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。与之对应的就是mounted钩子函数，因为该钩子函数执行时所有的DOM挂载已完成。  
   2、当项目中你想在改变DOM元素的数据后基于新的dom做点什么，对新DOM一系列的js操作都需要放进Vue.nextTick()的回调函数中；通俗的理解是：更改数据后当你想立即使用js操作新的视图的时候需要使用它
2. 在使用某个第三方插件时 ，希望在vue生成的某些dom动态发生变化时重新应用该插件，也会用到该方法，这时候就需要在 $nextTick 的回调函数中执行重新应用插件的方法。

**3. 你使用过 Vuex 吗？**

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 ( state )。

（1）Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

（2）改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。

主要包括以下几个模块：

State => 基本数据，定义了应用状态的数据结构，可以在这里设置默认的初始状态。

Getter => 从基本数据派生的数据，允许组件从 Store 中获取数据，mapGetters 辅助函数仅仅是将 store 中的 getter 映射到局部计算属性。

Mutation => 是唯一更改 store 中状态的方法，且必须是同步函数。

Action => 像一个装饰器，包裹mutations，使之可以异步。用于提交 mutation，而不是直接变更状态，可以包含任意异步操作。

Module => 模块化Vuex，允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中。

7. computed 和 watch 的区别和运用的场景？

computed： 是计算属性，依赖其它属性值，并且 computed 的值有缓存，只有它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值；

watch： 更多的是「观察」的作用，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；

运用场景：

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，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；

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

**9. 谈谈你对 Vue 生命周期的理解？**

生命周期是什么？

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，我们称这是 Vue 的生命周期。

各个生命周期的作用

| 生命周期 | 描述 |
| --- | --- |
| beforeCreate | 组件实例被创建之初，组件的属性生效之前 |
| 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，$el 还不可用 |
| beforeMount | 在挂载开始之前被调用：相关的 render 函数首次被调用 |
| mounted | el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子 |
| beforeUpdate | 组件数据更新之前调用，发生在虚拟 DOM 打补丁之前 |
| updated | 组件数据更新之后 |
| activited | keep-alive 专属，组件被激活时调用 |
| deadctivated | keep-alive 专属，组件被销毁时调用 |
| beforeDestory | 组件销毁前调用 |
| destoryed | 组件销毁后调用 |

**10. Vue 的父组件和子组件生命周期钩子函数执行顺序？**

Vue 的父组件和子组件生命周期钩子函数执行顺序可以归类为以下 4 部分：

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

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

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

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

**13. 组件中 data 为什么是一个函数？**

为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象？

因为组件是用来复用的，且 JS 里对象是引用关系，如果组件中 data 是一个对象，那么这样作用域没有隔离，子组件中的 data 属性值会相互影响，

如果组件中 data 选项是一个函数，那么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；而 new Vue 的实例，是不会被复用的，因此不存在引用对象的问题。

**15. Vue 组件间通信有哪几种方式？**

Vue 组件间通信是面试常考的知识点之一，这题有点类似于开放题，你回答出越多方法当然越加分，表明你对 Vue 掌握的越熟练。

Vue 组件间通信只要指以下 3 类通信：父子组件通信、隔代组件通信、兄弟组件通信，下面我们分别介绍每种通信方式且会说明此种方法可适用于哪类组件间通信。  
**（1）props / $emit 适用 父子组件通信**

这种方法是 Vue 组件的基础，相信大部分同学耳闻能详，所以此处就不举例展开介绍。

**（2）ref 与 $parent / $children适用 父子组件通信**

ref：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

$parent / $children：访问父 / 子实例

**（3）EventBus （$emit / $on）适用于 父子、隔代、兄弟组件通信**

这种方法通过一个空的 Vue 实例作为中央事件总线（事件中心），用它来触发事件和监听事件，从而实现任何组件间的通信，包括父子、隔代、兄弟组件。

**（4）$attrs/$listeners适用于 隔代组件通信**

$attrs：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 ( class 和 style 除外 )。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 ( class 和 style 除外 )，并且可以通过 v-bind="$attrs" 传入内部组件。通常配合 inheritAttrs 选项一起使用。

$listeners：包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件

**（5）provide / inject适用于 隔代组件通信**

祖先组件中通过 provider 来提供变量，然后在子孙组件中通过 inject 来注入变量。provide / inject API 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

**（6）Vuex适用于 父子、隔代、兄弟组件通信**

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 ( state )。

Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。

**17. vue-router 路由模式有几种？**

hash: 使用 URL hash 值来作路由。支持所有浏览器，包括不支持 HTML5 History Api 的浏览器；

history : 依赖 HTML5 History API 和服务器配置。具体可以查看 HTML5 History 模式；

abstract : 支持所有 JavaScript 运行环境，如 Node.js 服务器端。如果发现没有浏览器的 API，路由会自动强制进入这个模式.

**22. 虚拟 DOM 的优缺点？**

优点：

保证性能下限： 框架的虚拟 DOM 需要适配任何上层 API 可能产生的操作，它的一些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是比起粗暴的 DOM 操作性能要好很多，因此框架的虚拟 DOM 至少可以保证在你不需要手动优化的情况下，依然可以提供还不错的性能，即保证性能的下限；

无需手动操作 DOM： 我们不再需要手动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的方式更新视图，极大提高我们的开发效率；

跨平台： 虚拟 DOM 本质上是 JavaScript 对象,而 DOM 与平台强相关，相比之下虚拟 DOM 可以进行更方便地跨平台操作，例如服务器渲染、weex 开发等等。

缺点:

无法进行极致优化： 虽然虚拟 DOM + 合理的优化，足以应对绝大部分应用的性能需求，但在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化。

**23. 虚拟 DOM 实现原理？**

虚拟 DOM 的实现原理主要包括以下 3 部分：

用 JavaScript 对象模拟真实 DOM 树，对真实 DOM 进行抽象；

diff 算法 — 比较两棵虚拟 DOM 树的差异；

pach 算法 — 将两个虚拟 DOM 对象的差异应用到真正的 DOM 树。

24. Vue 中的 key 有什么作用？

key 是为 Vue 中 vnode 的唯一标记，通过这个 key，我们的 diff 操作可以更准确、更快速。

**25. 你有对 Vue 项目进行哪些优化？**

（1）代码层面的优化

v-if 和 v-show 区分使用场景

computed 和 watch 区分使用场景

v-for 遍历必须为 item 添加 key，且避免同时使用 v-if

长列表性能优化

事件的销毁

图片资源懒加载

路由懒加载

第三方插件的按需引入

优化无限列表性能

服务端渲染 SSR or 预渲染

（2）Webpack 层面的优化

Webpack 对图片进行压缩

减少 ES6 转为 ES5 的冗余代码

提取公共代码

模板预编译

提取组件的 CSS

优化 SourceMap

构建结果输出分析

Vue 项目的编译优化

（3）基础的 Web 技术的优化

开启 gzip 压缩

浏览器缓存

CDN 的使用

使用 Chrome Performance 查找性能瓶颈

**26. 对于 vue3.0 特性你有什么了解的吗？**

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

（1）监测机制的改变  
3.0 将带来基于代理 Proxy 的 observer 实现，提供全语言覆盖的反应性跟踪。这消除了 Vue 2 当中基于 Object.defineProperty 的实现所存在的很多限制：

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；

支持 Map、Set、WeakMap 和 WeakSet。

新的 observer 还提供了以下特性：

用于创建 observable 的公开 API。这为中小规模场景提供了简单轻量级的跨组件状态管理解决方案。

默认采用惰性观察。在 2.x 中，不管反应式数据有多大，都会在启动时被观察到。如果你的数据集很大，这可能会在应用启动时带来明显的开销。在 3.x 中，只观察用于渲染应用程序最初可见部分的数据。

更精确的变更通知。在 2.x 中，通过 Vue.set 强制添加新属性将导致依赖于该对象的 watcher 收到变更通知。在 3.x 中，只有依赖于特定属性的 watcher 才会收到通知。

不可变的 observable：我们可以创建值的“不可变”版本（即使是嵌套属性），除非系统在内部暂时将其“解禁”。这个机制可用于冻结 prop 传递或 Vuex 状态树以外的变化。

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）模板  
模板方面没有大的变更，只改了作用域插槽，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 TypeScript 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 TypeScript 的结合变得很容易。  
此外，vue 的源码也改用了 TypeScript 来写。其实当代码的功能复杂之后，必须有一个静态类型系统来做一些辅助管理。  
现在 vue3.0 也全面改用 TypeScript 来重写了，更是使得对外暴露的 api 更容易结合 TypeScript。静态类型系统对于复杂代码的维护确实很有必要。

（4）其它方面的更改  
vue3.0 的改变是全面的，上面只涉及到主要的 3 个方面，还有一些其他的更改：

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。

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。

基于 treeshaking 优化，提供了更多的内置功能。

1. **cookie,localStorage和sessionStorage的区别**

**webstorage本地存储**

**1）webstorage是本地存储，存储在客户端，包括localStorage和sessionStorage**

**2）localStorage生命周期是永久，这意味着除非用户显示在浏览器提供的UI上清除localStorage信息，否则这些信息将永远存在。存放数据大小为一般为5MB,而且它仅在客户端（即浏览器）中保存，不参与和服务器的通信**

**3）sessionStorage仅在当前会话下有效，关闭页面或浏览器后被清除。存放数据大小为一般为5MB,而且它仅在客户端（即浏览器）中保存，不参与和服务器的通信。源生接口可以接受，亦可再次封装来对Object和Array有更好的支持**

**4）WebStorage的目标**

**提供一种在cookie之外存储会话数据的路径**

**提供一种存储大量可以跨会话存在的数据的机制**

**HTML5的WebStorage提供了两种API：localStorage（本地存储）和sessionStorage（会话存储）**

**5）作用域的不同：**

**不同浏览器无法共享localStorage或sessionStorage中的信息。相同浏览器的不同页面间可以共享相同的 localStorage（页面属于相同域名和端口），但是不同页面或标签页间无法共享sessionStorage的信息。这里需要注意的是，页面及标 签页仅指顶级窗口，如果一个标签页包含多个iframe标签且他们属于同源页面，那么他们之间是可以共享sessionStorage的**

**6）存储大小：**

**localStorage和sessionStorage的存储数据大小一般都是：5MB**

**7）存储位置：**

**localStorage和sessionStorage都保存在客户端，不与服务器进行交互通信**

**8）存储内容类型：**

**localStorage和sessionStorage只能存储字符串类型，对于复杂的对象可以使用ECMAScript提供的JSON对象的stringify和parse来处理**

**9）获取方式：**

**localStorage：window.localStorage;；sessionStorage：window.sessionStorage;**

**10）应用场景：**

**localStoragese：常用于长期登录（+判断用户是否已登录），适合长期保存在本地的数据，而sessionStorage：敏感账号一次性登录**

**11）WebStorage的优点：**

**存储空间更大：cookie为4KB，而WebStorage是5MB**

**节省网络流量：WebStorage不会传送到服务器，存储在本地的数据可以直接获取，也不会像cookie一样美词请求都会传送到服务器，所以减少了客户端和服务器端的交互，节省了网络流量**

**对于那种只需要在用户浏览一组页面期间保存而关闭浏览器后就可以丢弃的数据，sessionStorage会非常方便**

**快速显示：有的数据存储在WebStorage上，再加上浏览器本身的缓存。获取数据时可以从本地获取会比从服务器端获取快得多，所以速度更快**

**安全性：WebStorage不会随着HTTP header发送到服务器端，所以安全性相对于cookie来说比较高一些，不会担心截获，但是仍然存在伪造问题**

**WebStorage提供了一些方法，数据操作比cookie方便**

**setItem (key, value) —— 保存数据，以键值对的方式储存信息。**

**getItem (key) —— 获取数据，将键值传入，即可获取到对应的value值。**

**removeItem (key) —— 删除单个数据，根据键值移除对应的信息。**

**clear () —— 删除所有的数据**

**key (index) —— 获取某个索引的key**

**cookie**

**1）HTTP Cookie简称cookie,在HTTP请求发送Set-Cookie HTTP头作为响应的一部分。通过name=value的形式存储**

**2）cookie的构成：**

**名称：name(不区分大小写,但最好认为它是区分的)**

**值:value(通过URL编码:encodeURIComponent)**

**域**

**路径**

**失效时间:一般默认是浏览器关闭失效,可以自己设置失效时间**

**安全标志:设置安全标志后只有SSL连接的时候才发送到服务器**

**3）cookie的作用:主要用于保存登录信息**

**4）生命期为只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。 存放数据大小为4K左右 。有个数限制（各浏览器不同），一般不能超过20个。与服务器端通信：每次都会携带在HTTP头中，如果使用cookie保存过多数据会带来性能问题**

**5）cookie的优点：具有极高的扩展性和可用性**

**通过良好的编程，控制保存在cookie中的session对象的大小**

**通过加密和安全传输技术，减少cookie被破解的可能性**

**只有在cookie中存放不敏感的数据，即使被盗取也不会有很大的损失**

**控制cookie的生命期，使之不会永远有效。这样的话偷盗者很可能拿到的就 是一个过期的cookie**

**6）cookie的缺点：**

**cookie的长度和数量的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB，否则会被截掉**

**安全性问题。如果cookie被人拦掉了，那个人就可以获取到所有session信息。加密的话也不起什么作用**

**有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务端保存一个计数器。若吧计数器保存在客户端，则起不到什么作用**

**sessionStorage**

**1）sessionStorage是Storage类型的一个对象，拥有clear(),getItem(name),key(index),removeItem(name),setItem(name,value)方法**

**2）sessionStorage对象存储特定于某个会话的数据,也就是该数据只保持到浏览器关闭**

**3）将数据保存在session对象中。所谓session，是指用户在浏览某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户浏览这个网站所花费的时间。session对象可以用来保存在这段时间内所要求保存的任何数据**

**4）sessionStorage为临时保存**

**localStorage**

**1）localStorage也是Storage类型的一个对象**

**2）在HTML5中localStorage作为持久保存在客户端数据的方案取代了globalStorage(globalStorage必须指定域名)**

**3）localStorage会永久存储会话数据，除非removeItem,否则会话数据一直存在**

**4）将数据保存在客户端本地的硬件设备(通常指硬盘，也可以是其他硬件设备)中，即使浏览器被关闭了，该数据仍然存在，下次打开浏览器访问网站时仍然可以继续使用**

**5）localStorage为永久保存**

**图示说明：**

**注意：时刻注意XSS注入的风险，因为可以在控制台直接访问它们，所以不要存入敏感数据**

**二、区别的比较**

**本地储存localStorage与cookie的区别**

**1）cookie在浏览器与服务器之间来回传递**

**sessionStorage和localStorage不会把数据发给服务器，仅在本地保存**

**2）数据有效期不同**

**cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭**

**sessionStorage：仅在当前浏览器窗口关闭前有效**

**localStorage 始终有效，长期保存**

**3）cookie数据还有路径的概念，可以限制cookie只属于某个路径下**

**存储大小也不同，cookie数据不能超过4k，sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大**

**4）作用域不用**

**sessionStorage不在不同的浏览器窗口中共享**

**localStorage在所有同源窗口中都是共享的**

**cookie也是在所有同源窗口中都是共享的**

**WebStorage 支持事件通知机制，可以将数据更新的通知发送给监听者。Web Storage 的 api 接口使用更方便**

**cookie、session和localStorage的区别**

**1）cookie的内容主要包括：名字、值、过期时间、路径和域，路径与域一起构成cookie的作用范围。若不设置时间，则表示这个cookie的生命期为浏览器会话期间，关闭浏览器窗口，cookie就会消失，这种生命期为浏览器会话期的cookie被称为会话cookie**

**2）会话cookie一般不存储在硬盘而是保存在内存里，当然这个行为并不是规范规定的。若设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再打开浏览器这些cookie仍然有效直到超过设定的过期时间。对于保存在内存里的cookie，不同的浏览器有不同的处理方式session机制。**

**3）当程序需要为某个客户端的请求创建一个session时，服务器首先检查这个客户端的请求里是否已包含了一个session标识（称为session id），如果已包含则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（检索不到，会新建一个），如果客户端请求不包含session id，则为客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。**

**cookie和session的区别**

**1）cookie数据存放在客户的浏览器上，session数据放在服务器上**

**2）cookie不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗，考虑到安全应当使用session**

**3）session会在一定时间内保存在服务器上，当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用cookie**

**4）单个cookie保存的数\*据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie**

**5）建议将登录信息等重要信息存放为session，其他信息如果需要保留，可以放在cookie中**

**6）session保存在服务器，客户端不知道其中的信心；cookie保存在客户端，服务器能够知道其中的信息**

**7）session中保存的是对象，cookie中保存的是字符串**

**8）session不能区分路径，同一个用户在访问一个网站期间，所有的session在任何一个地方都可以访问到，而cookie中如果设置了路径参数，那么同一个网站中不同路径下的cookie互相是访问不到的**

**web Storage和cookie的区别**

**1）Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的，cookie的大小是受限的，并且每次请求一个新的页面的时候cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可跨域调用**

**2）web storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie**

**3）但是cookie也是不可或缺的，cookie的作用是与服务器进行交互，作为http规范的一部分而存在的，而web Storage仅仅是为了在本地“存储”数据而生**

**sessionStorage、localStorage、cookie都是在浏览器端存储的数据，其中sessionStorage的概念很特别，引入了一个“浏览器窗口”的概念，sessionStorage是在同源的同窗口中，始终存在的数据，也就是说只要这个浏览器窗口没有关闭，即使刷新页面或进入同源另一个页面，数据仍然存在，关闭窗口后，sessionStorage就会被销毁，同时“独立”打开的不同窗口，即使是同一页面，sessionStorage对象也是不同的**

**4）Web Storage的好处**

**减少网络流量：一旦数据保存在本地之后，就可以避免再向服务器请求数据，因此减少不必要的数据请求，减少数据在浏览器和服务器间不必要的来回传递**

**快速显示数据：性能好，从本地读数据比通过网络从服务器上获得数据快得多，本地数据可以及时获得，再加上网页本身也可以有缓存，因此整个页面和数据都在本地的话，可以立即显示**

**临时存储：很多时候数据只需要在用户浏览一组页面期间使用，关闭窗口后数据就可以丢弃了，这种情况使用sessionStorage非常方便**

**浏览器本地存储与服务器端存储的区别**

**1）数据既可以在浏览器本地存储，也可以在服务器端存储**

**2）浏览器可以保存一些数据，需要的时候直接从本地存取，sessionStorage、localStorage和cookie都是由浏览器存储在本地的数据**

**3）服务器端也可以保存所有用户的所有数据，但需要的时候浏览器要向服务器请求数据**

**4）服务器端可以保存用户的持久数据，如数据库和云存储将用户的大量数据保存在服务器端 ，服务器端也可以保存用户的临时会话数据，服务器端的session机制，如jsp的session对象，数据保存在服务器上**

**5）服务器和浏览器之间仅需传递session id即可，服务器根据session id找到对应用户的session对象，会话数据仅在一段时间内有效，这个时间就是server端设置的session有效期**

**6）服务器端保存所有的用户的数据，所以服务器端的开销较大，而浏览器端保存则把不同用户需要的数据分别保存在用户各自的浏览器中，浏览器端一般只用来存储小数据，而非服务可以存储大数据或小数据服务器存储数据安全一些，浏览器只适合存储一般数据**

**sessionStorage、localStorage和cookie的区别**

**1）相同点是都是保存在浏览器端、且同源的**

**2）cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递，而sessionStorage和localStorage不会自动把数据发送给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下**

**3）存储大小限制也不同，cookie数据不能超过4K，同时因为每次http请求都会携带cookie、所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大**

**4）数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭之前有效；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie：只在设置的cookie过期时间之前有效，即使窗口关闭或浏览器关闭**

**5）作用域不同，sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面；localstorage在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的**

**6）web Storage支持事件通知机制，可以将数据更新的通知发送给监听者**

**7）web Storage的api接口使用更方便**

**sessionStorage与页面js数据对象的区别**

**1）页面中一般的js对象的生存期仅在当前页面有效，因此刷新页面或转到另一页面这样的重新加载页面的情况，数据就不存在了**

**2）sessionStorage只要同源的同窗口中，刷新页面或进入同源的不同页面，数据始终存在，也就是说只要浏览器不关闭，数据仍然存在**



**1.vue优点？**

答：

轻量级框架：只关注视图层，是一个构建数据的视图集合，大小只有几十 kb ；

简单易学：国人开发，中文文档，不存在语言障碍 ，易于理解和学习；

双向数据绑定：保留了 angular 的特点，在数据操作方面更为简单；

组件化：保留了 react 的优点，实现了 html 的封装和重用，在构建单页面应用方面有着独特的优势；

视图，数据，结构分离：使数据的更改更为简单，不需要进行逻辑代码的修改，只需要操作数据就能完成相关操作；

虚拟DOM：dom 操作是非常耗费性能的， 不再使用原生的 dom 操作节点，极大解放 dom 操作，但具体操作的还是 dom 不过是换了另一种方式；

运行速度更快：相比较于 react 而言，同样是操作虚拟 dom ，就性能而言， vue 存在很大的优势。

**2.vue父组件向子组件传递数据？**

答：通过 props

**3.子组件像父组件传递事件？**

答：$emit 方法

**4.v-show 和 v-if指令的共同点和不同点？**

答:

共同点： 都能控制元素的显示和隐藏；

不同点： 实现本质方法不同，v-show 本质就是通过控制 css 中的 display 设置为 none，控制隐藏，只会编译一次；v-if 是动态的向 DOM 树内添加或者删除 DOM 元素，若初始值为 false ，就不会编译了。而且 v-if 不停的销毁和创建比较消耗性能。总结：如果要频繁切换某节点，使用 v-show (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使用 v-if（初始渲染开销较小，切换开销比较大）。

**5. 如何让CSS只在当前组件中起作用?**

答：在组件中的 style 前面加上 scoped

6. <keep-alive></keep-alive> 的作用是什么?

答: 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。

**7.如何获取dom?**

答：ref="domName" 用法：this.$refs.domName

**8.说出几种vue当中的指令和它的用法？**

答：v-model 双向数据绑定；

v-for 循环；

v-if v-show 显示与隐藏；

v-on 事件；v-once : 只绑定一次。

**9. vue-loader是什么？使用它的用途有哪些？**

答：vue 文件的一个加载器，将 template/js/style 转换成 js 模块。

用途：js 可以写 es6 、 style样式可以 scss 或 less 、 template 可以加 jade等

**10.为什么使用key?**

答：需要使用 key 来给每个节点做一个唯一标识， Diff 算法就可以正确的识别此节点。作用主要是为了高效的更新虚拟 DOM。

**11.axios及安装?**

答：请求后台资源的模块。npm install axios —save 装好， js中使用 import 进来，然后 .get 或 .post 。返回在 .then 函数中如果成功，失败则是在 .catch 函数中。

**12.v-modal的使用**

答：v-model 用于表单数据的双向绑定，其实它就是一个语法糖，这个背后就做了两个操作：v-bind 绑定一个 value 属性；v-on 指令给当前元素绑定 input 事件。

**13.请说出vue.cli项目中src目录每个文件夹和文件的用法？**

答：assets 文件夹是放静态资源；components 是放组件；router 是定义路由相关的配置; app.vue 是一个应用主组件；main.js 是入口文件。

**14.分别简述computed和watch的使用场景**

答：

computed : 当一个属性受多个属性影响的时候就需要用到 computed ，最典型的栗子：购物车商品结算的时候

watch : 当一条数据影响多条数据的时候就需要用 watch ，栗子：搜索数据

**15.v-on可以监听多个方法吗？**

答：可以，栗子：<input type="text" v-on="{ input:onInput,focus:onFocus,blur:onBlur, }">。

**16.$nextTick的使用**

答：当你修改了data 的值然后马上获取这个 dom 元素的值，是不能获取到更新后的值， 你需要使用 $nextTick 这个回调，让修改后的 data 值渲染更新到 dom 元素之后在获取，才能成功。

**17.vue组件中data为什么必须是一个函数？**

答：因为 JavaScript 的特性所导致，在 component 中，data 必须以函数的形式存在，不可以是对象。组建中的 data 写成一个函数，数据以函数返回值的形式定义，这样每次复用组件的时候，都会返回一份新的 data ，相当于每个组件实例都有自己私有的数据空间，它们只负责各自维护的数据，不会造成混乱。而单纯的写成对象形式，就是所有的组件实例共用了一个 data ，这样改一个全都改了。

**18.渐进式框架的理解**

答：主张最少；可以根据不同的需求选择不同的层级；

**19.Vue中双向数据绑定是如何实现的？**

答：vue 双向数据绑定是通过 数据劫持 结合 发布订阅模式的方式来实现的， 也就是说数据和视图同步，数据发生变化，视图跟着变化，视图变化，数据也随之发生改变；核心：关于VUE双向数据绑定，其核心是 Object.defineProperty() 方法。

**20.单页面应用和多页面应用区别及优缺点**

答：单页面应用（SPA），通俗一点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的 html, js, css。所有的页面内容都包含在这个所谓的主页面中。但在写的时候，还是会分开写（页面片段），然后在交互的时候由路由程序动态载入，单页面的页面跳转，仅刷新局部资源。多应用于pc端。

多页面（MPA），就是指一个应用中有多个页面，页面跳转时是整页刷新

单页面的优点：用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点spa对服务器压力较小；前后端分离；页面效果会比较炫酷（比如切换页面内容时的专场动画）。

单页面缺点：不利于seo；导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）；初次加载时耗时多；页面复杂度提高很多。

**21.v-if和v-for的优先级**

答：当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。所以，不推荐 v-if 和 v-for 同时使用。如果 v-if 和 v-for 一起用的话，vue中的的会自动提示 v-if 应该放到外层去。

**22.assets和static的区别**

答：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，这是相同点

不相同点：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。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iconfoont.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，我们不再需要处理，直接上传。

**23.vue常用的修饰符**

答：

.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 ：与事件冒泡的方向相反，事件捕获由外到内；

.self ：只会触发自己范围内的事件，不包含子元素；

.once ：只会触发一次。

**24.vue的两个核心点**

答：数据驱动、组件系统

数据驱动： ViewModel，保证数据和视图的一致性。

组件系统： 应用类UI可以看作全部是由组件树构成的。

**25.vue和jQuery的区别**

答：jQuery是使用选择器（ $ ）选取DOM对象，对其进行赋值、取值、事件绑定等操作，其实和原生的HTML的区别只在于可以更方便的选取和操作DOM对象，而数据和界面是在一起的。比如需要获取label标签的内容：$("lable").val(); ,它还是依赖DOM元素的值。Vue则是通过Vue对象将数据和View完全分离开来了。对数据进行操作不再需要引用相应的DOM对象，可以说数据和View是分离的，他们通过Vue对象这个vm实现相互的绑定。这就是传说中的MVVM。

**26. 引进组件的步骤**

答: 在template中引入组件；在script的第一行用import引入路径；用component中写上组件名称。

27.delete和Vue.delete删除数组的区别

答：delete 只是被删除的元素变成了 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。

28.SPA首屏加载慢如何解决

答：安装动态懒加载所需插件；使用[CDN](https://cloud.tencent.com/product/cdn?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)资源。

**29.Vue-router跳转和location.href有什么区别**

答：使用 location.href= /url 来跳转，简单方便，但是刷新了页面；使用 history.pushState( /url ) ，无刷新页面，静态跳转；引进 router ，然后使用 router.push( /url ) 来跳转，使用了 diff 算法，实现了按需加载，减少了 dom 的消耗。其实使用 router 跳转和使用 history.pushState() 没什么差别的，因为vue-router就是用了 history.pushState() ，尤其是在history模式下。

**30. vue slot**

答：简单来说，假如父组件需要在子组件内放一些DOM，那么这些DOM是显示、不显示、在哪个地方显示、如何显示，就是slot分发负责的活。

**31.你们vue项目是打包了一个js文件，一个css文件，还是有多个文件？**

答：根据vue-cli脚手架规范，一个js文件，一个CSS文件。

**32.Vue里面router-link在电脑上有用，在安卓上没反应怎么解决？**

答：Vue路由在Android机上有问题，babel问题，安装babel polypill插件解决。

**33.Vue2中注册在router-link上事件无效解决方法**

答：使用 @click.native 。原因：router-link会阻止click事件，.native指直接监听一个原生事件。

34.RouterLink在IE和Firefox中不起作用（路由不跳转）的问题

答: 方法一：只用a标签，不适用button标签；方法二：使用button标签和Router.navigate方法

**35.axios的特点有哪些**

答：

从浏览器中创建XMLHttpRequests；

node.js创建http请求；

支持Promise API；

拦截请求和响应；

转换请求数据和响应数据；

取消请求；

自动换成json。

axios中的发送字段的参数是data跟params两个，两者的区别在于params是跟请求地址一起发送的，data的作为一个请求体进行发送

params一般适用于get请求，data一般适用于post put 请求。

**36.请说下封装 vue 组件的过程？**

答：

建立组件的模板，先把架子搭起来，写写样式，考虑好组件的基本逻辑。(os：思考1小时，码码10分钟，程序猿的准则。)

准备好组件的数据输入。即分析好逻辑，定好 props 里面的数据、类型。

准备好组件的数据输出。即根据组件逻辑，做好要暴露出来的方法。

封装完毕了，直接调用即可

**37.params和query的区别**

答：用法：query要用path来引入，params要用name来引入，接收参数都是类似的，分别是 this.$route.query.name 和 this.$route.params.name 。url地址显示：query更加类似于我们ajax中get传参，params则类似于post，说的再简单一点，前者在浏览器地址栏中显示参数，后者则不显示

注意点：query刷新不会丢失query里面的数据 params刷新 会 丢失 params里面的数据。

**38.vue初始化页面闪动问题**

答：使用vue开发时，在vue初始化之前，由于 div 是不归 vue 管的，所以我们写的代码在还没有解析的情况下会容易出现花屏现象，看到类似于 {{message}} 的字样，虽然一般情况下这个时间很短暂，但是我们还是有必要让解决这个问题的。首先：在css里加上 [v-cloak] { display: none; } 。如果没有彻底解决问题，则在根元素加上 style="display: none;" :style="{display: block }"

**39.vue更新数组时触发视图更新的方法**

答:push()；pop()；shift()；unshift()；splice()；sort()；reverse()

**40.vue常用的UI组件库**

答：Mint UI，element，VUX

**41.vue修改打包后静态资源路径的修改**

答：cli2 版本：将 config/index.js 里的 assetsPublicPath 的值改为 ./ 。

build: {

// ...

assetsPublicPath: ./ ,

// ... }

cli3版本：在根目录下新建vue.config.js 文件，然后加上以下内容：（如果已经有此文件就直接修改）

module.exports = {

publicPath: , // 相对于 HTML 页面（目录相同） }

生命周期函数面试题篇

### **42.什么是 vue 生命周期？有什么作用？**

答：每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做 生命周期钩子 的函数，这给了用户在不同阶段添加自己的代码的机会。（ps：生命周期钩子就是生命周期函数）例如，如果要通过某些插件操作DOM节点，如想在页面渲染完后弹出广告窗， 那我们最早可在mounted 中进行。

**43.第一次页面加载会触发哪几个钩子？**

答：beforeCreate， created， beforeMount， mounted

**44.简述每个周期具体适合哪些场景**

答：

beforeCreate： 在new一个vue实例后，只有一些默认的生命周期钩子和默认事件，其他的东西都还没创建。在beforeCreate生命周期执行的时候，data和methods中的数据都还没有初始化。不能在这个阶段使用data中的数据和methods中的方法

create： data 和 methods都已经被初始化好了，如果要调用 methods 中的方法，或者操作 data 中的数据，最早可以在这个阶段中操作

beforeMount： 执行到这个钩子的时候，在内存中已经编译好了模板了，但是还没有挂载到页面中，此时，页面还是旧的

mounted： 执行到这个钩子的时候，就表示Vue实例已经初始化完成了。此时组件脱离了创建阶段，进入到了运行阶段。如果我们想要通过插件操作页面上的DOM节点，最早可以在和这个阶段中进行

beforeUpdate： 当执行这个钩子时，页面中的显示的数据还是旧的，data中的数据是更新后的， 页面还没有和最新的数据保持同步

updated： 页面显示的数据和data中的数据已经保持同步了，都是最新的

beforeDestory： Vue实例从运行阶段进入到了销毁阶段，这个时候上所有的 data 和 methods ， 指令， 过滤器 ……都是处于可用状态。还没有真正被销毁

destroyed： 这个时候上所有的 data 和 methods ， 指令， 过滤器 ……都是处于不可用状态。组件已经被销毁了。

**45.created和mounted的区别**

答：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。

**46.vue获取数据在哪个周期函数**

答：一般 created/beforeMount/mounted 皆可. 比如如果你要操作 DOM , 那肯定 mounted 时候才能操作.

**47.请详细说下你对vue生命周期的理解？**

答：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。

vue 路由面试题篇

**48.mvvm 框架是什么？**

答：vue是实现了双向数据绑定的mvvm框架，当视图改变更新模型层，当模型层改变更新视图层。在vue中，使用了双向绑定技术，就是View的变化能实时让Model发生变化，而Model的变化也能实时更新到View。

**49.vue-router 是什么?它有哪些组件**

答：vue用来写路由一个插件。router-link、router-view

**50.active-class 是哪个组件的属性？**

答：vue-router模块的router-link组件。children数组来定义子路由

**51.怎么定义 vue-router 的动态路由? 怎么获取传过来的值？**

答：在router目录下的index.js文件中，对path属性加上/:id。使用router对象的params.id。

**52.vue-router 有哪几种导航钩子?**

答：三种，

第一种： 是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。

第二种： 组件内的钩子

第三种： 单独路由独享组件

**53. `$route` 和 `$router` 的区别**

答：$router 是VueRouter的实例，在script标签中想要导航到不同的URL,使用 $router.push 方法。返回上一个历史history用 $router.to(-1)

$route 为当前router跳转对象。里面可以获取当前路由的name,path,query,parmas等。

**54.vue-router的两种模式**

答: hash模式： 即地址栏 URL 中的 # 符号；

history模式： window.history对象打印出来可以看到里边提供的方法和记录长度。利用了 HTML5 History Interface 中新增的 pushState() 和 replaceState() 方法。（需要特定浏览器支持）。

**55.vue-router实现路由懒加载（ 动态加载路由 ）**

答:三种方式

第一种： vue异步组件技术 ==== 异步加载，vue-router配置路由 , 使用vue的异步组件技术 , 可以实现按需加载 .但是,这种情况下一个组件生成一个js文件。

第二种： 路由懒加载(使用import)。

第三种： webpack提供的require.ensure()，vue-router配置路由，使用webpack的require.ensure技术，也可以实现按需加载。这种情况下，多个路由指定相同的chunkName，会合并打包成一个js文件。

vuex常见面试题篇

**56.vuex是什么？怎么使用？哪种功能场景使用它？**

答：vue框架中状态管理。在main.js引入store，注入。

新建了一个目录store.js，….. export 。

场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

**57.vuex有哪几种属性？**

答：有五种，分别是 State、 Getter、Mutation 、Action、 Module

state => 基本数据(数据源存放地)

getters => 从基本数据派生出来的数据

mutations => 提交更改数据的方法，同步！

actions => 像一个装饰器，包裹mutations，使之可以异步。

modules => 模块化Vuex

1. **React**

Vue和react区别

监听数据变化的实现原理不同

数据流的不同

HoC和mixins

组件通信的区别

模板渲染方式的不同

渲染过程不同

框架本质不同

Vuex和Redux的区别

1. **笔试题**
2. 递归
3. 求阶乘

//从自身往前递推并将所递推的数相乘，结果用于下一次计算

function jiecheng(num){

//找到出口

if(num<=1){

return 1

} else{

return num\*jiecheng(num-1)

} }

console.log(jiecheng(4));

2.递归实现多层对象嵌套的深拷贝

const deepClone=(obj)=>{

let temp={}

//遍历对象，判断对象中属性值是否是对象，如果是重复操作

for(let k in obj){

if(typeof obj[k]=='object'){

temp[k]=deepClone(obj[k]);

}

else{

temp[k]=obj[k]

}

}

return temp

}

3 树递归遍历取值

getLeafkey(node,arr){

if(!node.children){

return arr.push(node.id)

}

node.children.forEach(item=>{

this. getLeafkey(item,arr)

})

}

//这里的node表示所有的节点，如果该节点上没有children属性，就表示已经到达三级节点，如果存在则继续遍历数组，并递归这一级

4 递归实现扁平化

需求描述： const o = { a: 1, b: [1, 2, { c: true }], d: { e: 2, f: 3 }, g: null };fn(o) => 扁平化转换{ "a": 1, "b[0]": 1, "b[1]": 2, "b[2].c": true, "d.e": 2, ... }

const o = { a: 1, b: [1, 2, { c: true }], d: { e: 2, f: 3 }, g: null };

const fn = (root) => {

const obj = {};

const f1 = (root, propName) => {

//循环对象，并迭代成数组，键和值形成数组，并存放在一个大数组内

for (let [key, value] of Object.entries(root)) {

//判断这个值是否存在并类型是否是一个对象

if (value && typeof value === "object") {

//是一个对象，则把这个对象递归上面一步，将这个对象在进行调用传值，

//根据对象中得值是否是一个数组判断proName是否存在，

//如果存在，将该变量跟当前得对象一起传入函数进行递归

f1(value, propName ? `${propName}[${key}]` : key);

} else {

//这里proName存在，此时得root应该是个数组

if (propName) {

if (Array.isArray(root)) {

//这里相当于b[0]:1,b[1]:2

obj[`${propName}[${key}]`] = value;

} else {

obj[`${propName}.${key}`] = value;

}

} else {

//这里相当于实现"b[2].c": true

obj[key] = value;

}

}

}

};

f1(root);

return obj;

};

console.log(fn(o));

5 递归实现斐波那契数列

斐波那契数列是一系列数字：0、1、1、2、3、5、8、13、21、34 等等。该模式涉及将前两个数字相加，因此 0 + 1 = 1、1 + 1 = 2、1 + 2 = 3、2 + 3 = 5 等。换句话说，位置n(for n > 2) 处的斐波那契数是斐波那契数的(n - 1)加上斐波那契的(n - 2)。

function fibonacciSequence(n) {

//先定义第一项和第二项

const memo = [0, 1];

const fib = (n) => {

//当数组到

if (memo[n] != null) return memo[n];

//后面一个数等于数组前两项之和

return memo[n] = fib(n - 1, memo) + fib(n - 2, memo);

};

return fib(n);

}

6 递归计算数组索引

需求：给定一个数字数组，找到第一个负值的索引

let sals = [134.56, 562.22, 357.81, -303.73, 256.45, -453.78];

const debitIndex = (data, index) => {

if (index === data.length) {

return false

}

else if (data[index] < 0) {

return index

}

else{

return debitIndex(data, index + 1)

};

};

let result = debitIndex(sals, 0);

1. 数组去重

JS数组去重的方式

例：将下面数组去除重复元素（以多种数据类型为例）

const arr = [1, 2, 2, 'abc', 'abc', true, true, false, false, undefined, undefined, NaN, NaN]

1.利用Set()+Array.from()

Set对象：是值的集合，你可以按照插入的顺序迭代它的元素。 Set中的元素只会出现一次，即Set中的元素是唯一的。

Array.from() 方法：对一个类似数组或可迭代对象创建一个新的，浅拷贝的数组实例。

const result = Array.from(new Set(arr))

console.log(result) // [ 1, 2, 'abc', true, false, undefined, NaN ]

注意：以上去方式对NaN和undefined类型去重也是有效的，是因为NaN和undefined都可以被存储在Set中， NaN之间被视为相同的值（尽管在js中：NaN !== NaN）。

2.利用两层循环+数组的splice方法

通过两层循环对数组元素进行逐一比较，然后通过splice方法来删除重复的元素。此方法对NaN是无法进行去重的，因为进行比较时NaN !== NaN。

function removeDuplicate(arr) {

let len = arr.length

for (let i = 0; i < len; i++) {

for (let j = i + 1; j < len; j++) {

if (arr[i] === arr[j]) {

arr.splice(j, 1)

len-- // 减少循环次数提高性能

j-- // 保证j的值自加后不变

}

}

}

return arr

}

const result = removeDuplicate(arr)

console.log(result) // [ 1, 2, 'abc', true, false, undefined, NaN, NaN ]

3.利用数组的indexOf方法

新建一个空数组，遍历需要去重的数组，将数组元素存入新数组中，存放前判断数组中是否已经含有当前元素，没有则存入。此方法也无法对NaN去重。

indexOf() 方法：返回调用它的String对象中第一次出现的指定值的索引，从 fromIndex 处进行搜索。如果未找到该值，则返回 -1。

function removeDuplicate(arr) {

const newArr = []

arr.forEach(item => {

if (newArr.indexOf(item) === -1) {

newArr.push(item)

}

})

return newArr // 返回一个新数组

}

const result = removeDuplicate(arr)

console.log(result) // [ 1, 2, 'abc', true, false, undefined, NaN, NaN ]

4.利用数组的includes方法

此方法逻辑与indexOf方法去重异曲同工，只是用includes方法来判断是否包含重复元素。

includes()方法：用来判断一个数组是否包含一个指定的值，根据情况，如果包含则返回 true，否则返回 false。

function removeDuplicate(arr) {

const newArr = []

arr.forEach(item => {

if (!newArr.includes(item)) {

newArr.push(item)

}

})

return newArr

}

const result = removeDuplicate(arr)

console.log(result) // [ 1, 2, 'abc', true, false, undefined, NaN ]

注意：为什么includes能够检测到数组中包含NaN，其涉及到includes底层的实现。如下图为includes实现的部分代码，在进行判断是否包含某元素时会调用sameValueZero方法进行比较，如果为NaN，则会使用isNaN()进行转化。

具体实现可参考：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global\_Objects/Array/includes

简单测试includes()对NaN的判断：

const testArr = [1, 'a', NaN]

console.log(testArr.includes(NaN)) // true

5.利用数组的filter()+indexOf()

filter方法会对满足条件的元素存放到一个新数组中，结合indexOf方法进行判断。

filter() 方法：会创建一个新数组，其包含通过所提供函数实现的测试的所有元素。

function removeDuplicate(arr) {

return arr.filter((item, index) => {

return arr.indexOf(item) === index

})

}

const result = removeDuplicate(arr)

console.log(result) // [ 1, 2, 'abc', true, false, undefined ]

注意：这里的输出结果中不包含NaN，是因为indexOf()无法对NaN进行判断，即arr.indexOf(item) === index返回结果为false。测试如下：

const testArr = [1, 'a', NaN]

console.log(testArr.indexOf(NaN)) // -1

6.利用Map()

Map对象是JavaScript提供的一种数据结构，结构为键值对形式，将数组元素作为map的键存入，然后结合has()和set()方法判断键是否重复。

Map 对象：用于保存键值对，并且能够记住键的原始插入顺序。任何值（对象或者原始值）都可以作为一个键或一个值。

function removeDuplicate(arr) {

const map = new Map()

const newArr = []

arr.forEach(item => {

if (!map.has(item)) { // has()用于判断map是否包为item的属性值

map.set(item, true) // 使用set()将item设置到map中，并设置其属性值为true

newArr.push(item)

}

})

return newArr

}

const result = removeDuplicate(arr)

console.log(result) // [ 1, 2, 'abc', true, false, undefined, NaN ]

注意：使用Map()也可对NaN去重，原因是Map进行判断时认为NaN是与NaN相等的，剩下所有其它的值是根据 === 运算符的结果判断是否相等。

7.利用对象

其实现思想和Map()是差不多的，主要是利用了对象的属性名不可重复这一特性。

function removeDuplicate(arr) {

const newArr = []

const obj = {}

arr.forEach(item => {

if (!obj[item]) {

newArr.push(item)

obj[item] = true

}

})

return newArr

}

const result = removeDuplicate(arr)

console.log(result) // [ 1, 2, 'abc', true, false, undefined, NaN ]

1. 冒泡排序

var arr = [3, 10, 6, 2];// 遍历数组

for (var i = 0; i < arr.length - 1; i++) {

// 这里要根据外层for循环的 i ，逐渐减少内层 for 循环的次数

for (var j = 0; j < arr.length - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

var num = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = num;

}

}}

console.log(arr) // [ 2, 3, 6, 10 ]

### 1. HTTP 和 HTTPS

#### 1.http 和 https 的基本概念

http: 是一个客户端和服务器端请求和应答的标准（TCP），用于从 WWW 服务器传输超文本到本地浏览器的超文本传输协议。  
https:是以安全为目标的 HTTP 通道，即 HTTP 下 加入 SSL 层进行加密。其作用是：建立一个信息安全通道，来确保数据的传输，确保网站的真实性。

#### 2.http 和 https 的区别及优缺点？

* http 是超文本传输协议，信息是明文传输，HTTPS 协议要比 http 协议安全，https 是具有安全性的 ssl 加密传输协议，可防止数据在传输过程中被窃取、改变，确保数据的完整性(当然这种安全性并非绝对的，对于更深入的 Web 安全问题，此处暂且不表)。
* http 协议的默认端口为 80，https 的默认端口为 443。
* http 的连接很简单，是无状态的。https 握手阶段比较费时，会使页面加载时间延长 50%，增加 10%~20%的耗电。
* https 缓存不如 http 高效，会增加数据开销。
* Https 协议需要 ca 证书，费用较高，功能越强大的证书费用越高。
* SSL 证书需要绑定 IP，不能再同一个 IP 上绑定多个域名，IPV4 资源支持不了这种消耗。

#### 3.https 协议的工作原理

客户端在使用 HTTPS 方式与 Web 服务器通信时有以下几个步骤：

1. 客户端使用 https url 访问服务器，则要求 web 服务器建立 ssl 链接。
2. web 服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），传输给客户端。
3. 客户端和 web 服务器端开始协商 SSL 链接的安全等级，也就是加密等级。
4. 客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。
5. web 服务器通过自己的私钥解密出会话密钥。
6. web 服务器通过会话密钥加密与客户端之间的通信。

传送门 ☞ [# 解读 HTTP1/HTTP2/HTTP3](https://juejin.cn/post/6995109407545622542" \o "https://juejin.cn/post/6995109407545622542" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### TCP三次握手

1. 第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入SYN\_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。
2. 第二次握手：服务器收到syn包并确认客户的SYN（ack=j+1），同时也发送一个自己的SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；
3. 第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。

### TCP 四次挥手

1. 客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。 TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。

2）服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。

3）客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最 后的数据）。

4）服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认。

5）客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2∗∗MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。

6）服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

### TCP/IP / 如何保证数据包传输的有序可靠？

对字节流分段并进行编号然后通过 ACK 回复和超时重发这两个机制来保证。

（1）为了保证数据包的可靠传递，发送方必须把已发送的数据包保留在缓冲区；  
（2）并为每个已发送的数据包启动一个超时定时器；  
（3）如在定时器超时之前收到了对方发来的应答信息（可能是对本包的应答，也可以是对本包后续包的应答），则释放该数据包占用的缓冲区;  
（4）否则，重传该数据包，直到收到应答或重传次数超过规定的最大次数为止。  
（5）接收方收到数据包后，先进行CRC校验，如果正确则把数据交给上层协议，然后给发送方发送一个累计应答包，表明该数据已收到，如果接收方正好也有数据要发给发送方，应答包也可方在数据包中捎带过去。

### TCP和UDP的区别

1. TCP是面向链接的，而UDP是面向无连接的。
2. TCP仅支持单播传输，UDP 提供了单播，多播，广播的功能。
3. TCP的三次握手保证了连接的可靠性; UDP是无连接的、不可靠的一种数据传输协议，首先不可靠性体现在无连接上，通信都不需要建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收。
4. UDP的头部开销比TCP的更小，数据传输速率更高，实时性更好。

传送门 ☞ [# 深度剖析TCP与UDP的区别](https://juejin.cn/post/6992743999756845087" \o "https://juejin.cn/post/6992743999756845087" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### HTTP 请求跨域问题

跨域的原理

**跨域**，是指浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的。  
**同源策略**,是浏览器对 JavaScript 实施的安全限制，只要协议、域名、端口有任何一个不同，都被当作是不同的域。  
**跨域原理**，即是通过各种方式，避开浏览器的安全限制。

解决方案

最初做项目的时候，使用的是jsonp，但存在一些问题，使用get请求不安全，携带数据较小，后来也用过iframe，但只有主域相同才行，也是存在些问题，后来通过了解和学习发现使用代理和proxy代理配合起来使用比较方便，就引导后台按这种方式做下服务器配置，在开发中使用proxy，在服务器上使用nginx代理，这样开发过程中彼此都方便，效率也高；现在h5新特性还有 windows.postMessage()

**JSONP**：  
ajax 请求受同源策略影响，不允许进行跨域请求，而 script 标签 src 属性中的链 接却可以访问跨域的 js 脚本，利用这个特性，服务端不再返回 JSON 格式的数据，而是 返回一段调用某个函数的 js 代码，在 src 中进行了调用，这样实现了跨域。

步骤：

* + 1. 去创建一个script标签
    2. script的src属性设置接口地址
    3. 接口参数，必须要带一个自定义函数名，要不然后台无法返回数据
    4. 通过定义函数名去接受返回的数据

//动态创建 scriptvar script = document.createElement('script');

// 设置回调函数function getData(data) {

console.log(data);

}

//设置 script 的 src 属性，并设置请求地址

script.src = 'http://localhost:3000/?callback=getData';

// 让 script 生效document.body.appendChild(script);复制代码

**JSONP 的缺点**:  
JSON 只支持 get，因为 script 标签只能使用 get 请求； JSONP 需要后端配合返回指定格式的数据。

**document.domain** 基础域名相同 子域名不同

**window.name** 利用在一个浏览器窗口内，载入所有的域名都是共享一个window.name

**CORS** CORS(Cross-origin resource sharing)跨域资源共享 服务器设置对CORS的支持原理：服务器设置Access-Control-Allow-Origin HTTP响应头之后，浏览器将会允许跨域请求

**proxy代理** 目前常用方式,通过服务器设置代理

**window.postMessage()** 利用h5新特性window.postMessage()

跨域传送门 ☞ [# 跨域，不可不知的基础概念](https://juejin.cn/post/7003232769182547998" \o "https://juejin.cn/post/7003232769182547998" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### Cookie、sessionStorage、localStorage 的区别

**相同点**：

* 存储在客户端

**不同点**：

* cookie数据大小不能超过4k；sessionStorage和localStorage的存储比cookie大得多，可以达到5M+
* cookie设置的过期时间之前一直有效；localStorage永久存储，浏览器关闭后数据不丢失除非主动删除数据；sessionStorage数据在当前浏览器窗口关闭后自动删除
* cookie的数据会自动的传递到服务器；sessionStorage和localStorage数据保存在本地

### 粘包问题分析与对策

TCP粘包是指发送方发送的若干包数据到接收方接收时粘成一包，从接收缓冲区看，后一包数据的头紧接着前一包数据的尾。

**粘包出现原因**

简单得说，在流传输中出现，UDP不会出现粘包，因为它有**消息边界**

粘包情况有两种，一种是粘在一起的包都是完整的数据包，另一种情况是粘在一起的包有不完整的包。

为了**避免粘包**现象，可采取以下几种措施：

（1）对于发送方引起的粘包现象，用户可通过编程设置来避免，TCP提供了强制数据立即传送的操作指令push，TCP软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；

（2）对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；

（3）由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。分包多发。

以上提到的三种措施，都有其不足之处。

（1）第一种编程设置方法虽然可以避免发送方引起的粘包，但它关闭了优化算法，降低了网络发送效率，影响应用程序的性能，一般不建议使用。

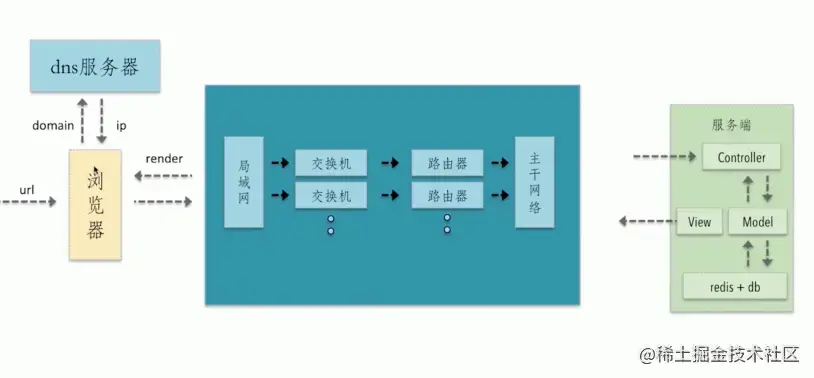
（2）第二种方法只能减少出现粘包的可能性，但并不能完全避免粘包，当发送频率较高时，或由于网络突发可能使某个时间段数据包到达接收方较快，接收方还是有可能来不及接收，从而导致粘包。

（3）第三种方法虽然避免了粘包，但应用程序的效率较低，对实时应用的场合不适合。

一种比较周全的对策是：接收方创建一预处理线程，对接收到的数据包进行预处理，将粘连的包分开。实验证明这种方法是高效可行的。

## 浏览器

### 从输入URL到页面加载的全过程



首先在浏览器中输入URL

查找缓存：浏览器先查看浏览器缓存-系统缓存-路由缓存中是否有该地址页面，如果有则显示页面内容。如果没有则进行下一步。

* + 浏览器缓存：浏览器会记录DNS一段时间，因此，只是第一个地方解析DNS请求；
  + 操作系统缓存:如果在浏览器缓存中不包含这个记录，则会使系统调用操作系统， 获取操作系统的记录(保存最近的DNS查询缓存)；
  + 路由器缓存：如果上述两个步骤均不能成功获取DNS记录，继续搜索路由器缓存；
  + ISP缓存：若上述均失败，继续向ISP搜索。

DNS域名解析：浏览器向DNS服务器发起请求，解析该URL中的域名对应的IP地址。DNS服务器是基于UDP的，因此会用到UDP协议。

建立TCP连接：解析出IP地址后，根据IP地址和默认80端口，和服务器建立TCP连接

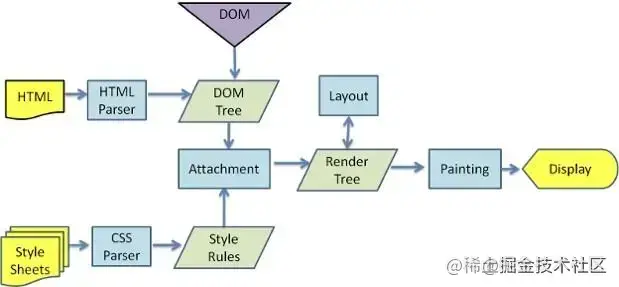
发起HTTP请求：浏览器发起读取文件的HTTP请求，，该请求报文作为TCP三次握手的第三次数据发送给服务器

服务器响应请求并返回结果：服务器对浏览器请求做出响应，并把对应的html文件发送给浏览器

关闭TCP连接：通过四次挥手释放TCP连接

浏览器渲染：客户端（浏览器）解析HTML内容并渲染出来，浏览器接收到数据包后的解析流程为：

* + 构建DOM树：词法分析然后解析成DOM树（dom tree），是由dom元素及属性节点组成，树的根是document对象
  + 构建CSS规则树：生成CSS规则树（CSS Rule Tree）
  + 构建render树：Web浏览器将DOM和CSSOM结合，并构建出渲染树（render tree）
  + 布局（Layout）：计算出每个节点在屏幕中的位置
  + 绘制（Painting）：即遍历render树，并使用UI后端层绘制每个节点。



JS引擎解析过程：调用JS引擎执行JS代码（JS的解释阶段，预处理阶段，执行阶段生成执行上下文，VO，作用域链、回收机制等等）

* + 创建window对象：window对象也叫全局执行环境，当页面产生时就被创建，所有的全局变量和函数都属于window的属性和方法，而DOM Tree也会映射在window的doucment对象上。当关闭网页或者关闭浏览器时，全局执行环境会被销毁。
  + 加载文件：完成js引擎分析它的语法与词法是否合法，如果合法进入预编译
  + 预编译：在预编译的过程中，浏览器会寻找全局变量声明，把它作为window的属性加入到window对象中，并给变量赋值为'undefined'；寻找全局函数声明，把它作为window的方法加入到window对象中，并将函数体赋值给他（匿名函数是不参与预编译的，因为它是变量）。而变量提升作为不合理的地方在ES6中已经解决了，函数提升还存在。
  + 解释执行：执行到变量就赋值，如果变量没有被定义，也就没有被预编译直接赋值，在ES5非严格模式下这个变量会成为window的一个属性，也就是成为全局变量。string、int这样的值就是直接把值放在变量的存储空间里，object对象就是把指针指向变量的存储空间。函数执行，就将函数的环境推入一个环境的栈中，执行完成后再弹出，控制权交还给之前的环境。JS作用域其实就是这样的执行流机制实现的。

传送门 ☞ [# DNS域名解析过程](https://juejin.cn/post/7005468491067162655" \o "https://juejin.cn/post/7005468491067162655" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank) ☞[# 浏览器的工作原理](https://juejin.cn/post/6992597760935460901" \o "https://juejin.cn/post/6992597760935460901" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### 浏览器重绘与重排的区别？

* 重排/回流（Reflow）：当DOM的变化影响了元素的几何信息，浏览器需要重新计算元素的几何属性，将其安放在界面中的正确位置，这个过程叫做重排。表现为重新生成布局，重新排列元素。
* 重绘(Repaint): 当一个元素的外观发生改变，但没有改变布局,重新把元素外观绘制出来的过程，叫做重绘。表现为某些元素的外观被改变

单单改变元素的外观，肯定不会引起网页重新生成布局，但当浏览器完成重排之后，将会重新绘制受到此次重排影响的部分

重排和重绘代价是高昂的，它们会破坏用户体验，并且让UI展示非常迟缓，而相比之下重排的性能影响更大，在两者无法避免的情况下，一般我们宁可选择代价更小的重绘。

『重绘』不一定会出现『重排』，『重排』必然会出现『重绘』。

### 如何触发重排和重绘？

任何改变用来构建渲染树的信息都会导致一次重排或重绘：

* 添加、删除、更新DOM节点
* 通过display: none隐藏一个DOM节点-触发重排和重绘
* 通过visibility: hidden隐藏一个DOM节点-只触发重绘，因为没有几何变化
* 移动或者给页面中的DOM节点添加动画
* 添加一个样式表，调整样式属性
* 用户行为，例如调整窗口大小，改变字号，或者滚动。

### 如何避免重绘或者重排？

集中改变样式，不要一条一条地修改 DOM 的样式。

不要把 DOM 结点的属性值放在循环里当成循环里的变量。

为动画的 HTML 元件使用 fixed 或 absoult 的 position，那么修改他们的 CSS 是不会 reflow 的。

不使用 table 布局。因为可能很小的一个小改动会造成整个 table 的重新布局。

尽量只修改position：absolute或fixed元素，对其他元素影响不大

动画开始GPU加速，translate使用3D变化

提升为合成层

将元素提升为合成层有以下优点：

* + 合成层的位图，会交由 GPU 合成，比 CPU 处理要快
  + 当需要 repaint 时，只需要 repaint 本身，不会影响到其他的层
  + 对于 transform 和 opacity 效果，不会触发 layout 和 paint

提升合成层的最好方式是使用 CSS 的 will-change 属性：

#target {

will-change: transform;

}复制代码

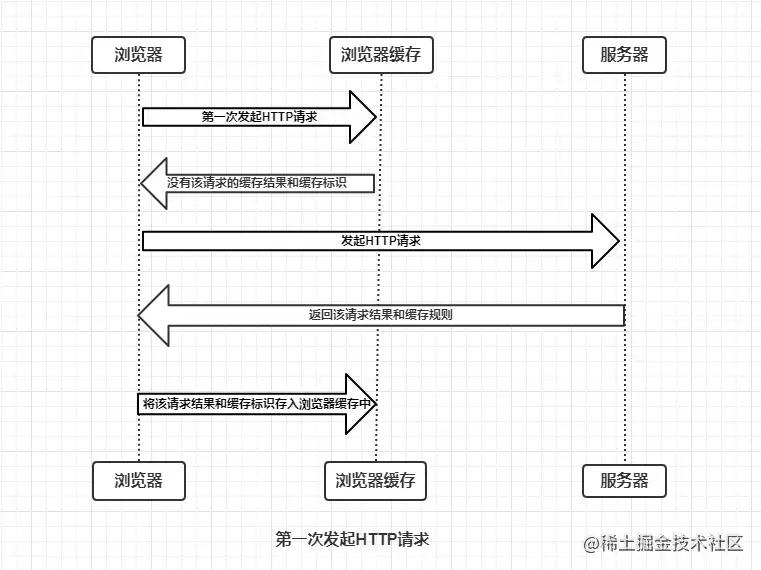
关于合成层的详解请移步[无线性能优化：Composite](https://link.juejin.cn?target=http://taobaofed.org/blog/2016/04/25/performance-composite/" \o "https://link.juejin.cn?target=http%3A%2F%2Ftaobaofed.org%2Fblog%2F2016%2F04%2F25%2Fperformance-composite%2F" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### 介绍下304过程

* a. 浏览器请求资源时首先命中资源的Expires 和 Cache-Control，Expires 受限于本地时间，如果修改了本地时间，可能会造成缓存失效，可以通过Cache-control: max-age指定最大生命周期，状态仍然返回200，但不会请求数据，在浏览器中能明显看到from cache字样。
* b. 强缓存失效，进入协商缓存阶段，首先验证ETagETag可以保证每一个资源是唯一的，资源变化都会导致ETag变化。服务器根据客户端上送的If-None-Match值来判断是否命中缓存。
* c. 协商缓存Last-Modify/If-Modify-Since阶段，客户端第一次请求资源时，服务服返回的header中会加上Last-Modify，Last-modify是一个时间标识该资源的最后修改时间。再次请求该资源时，request的请求头中会包含If-Modify-Since，该值为缓存之前返回的Last-Modify。服务器收到If-Modify-Since后，根据资源的最后修改时间判断是否命中缓存。

### 浏览器的缓存机制 强制缓存 && 协商缓存

浏览器与服务器通信的方式为应答模式，即是：浏览器发起HTTP请求 – 服务器响应该请求。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，简单的过程如下图：



由上图我们可以知道：

* 浏览器每次发起请求，都会先在浏览器缓存中查找该请求的结果以及缓存标识
* 浏览器每次拿到返回的请求结果都会将该结果和缓存标识存入浏览器缓存中

以上两点结论就是浏览器缓存机制的关键，他确保了每个请求的缓存存入与读取，只要我们再理解浏览器缓存的使用规则，那么所有的问题就迎刃而解了。为了方便理解，这里根据是否需要向服务器重新发起HTTP请求将缓存过程分为两个部分，分别是强制缓存和协商缓存。

**强制缓存**

强制缓存就是向浏览器缓存查找该请求结果，并根据该结果的缓存规则来决定是否使用该缓存结果的过程。当浏览器向服务器发起请求时，服务器会将缓存规则放入HTTP响应报文的HTTP头中和请求结果一起返回给浏览器，控制强制缓存的字段分别是 Expires 和 Cache-Control，其中Cache-Control优先级比Expires高。

强制缓存的情况主要有三种(暂不分析协商缓存过程)，如下：

* 1. 不存在该缓存结果和缓存标识，强制缓存失效，则直接向服务器发起请求（跟第一次发起请求一致）。
  2. 存在该缓存结果和缓存标识，但该结果已失效，强制缓存失效，则使用协商缓存。
  3. 存在该缓存结果和缓存标识，且该结果尚未失效，强制缓存生效，直接返回该结果

**协商缓存**

协商缓存就是强制缓存失效后，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程，同样，协商缓存的标识也是在响应报文的HTTP头中和请求结果一起返回给浏览器的，控制协商缓存的字段分别有：Last-Modified / If-Modified-Since 和 Etag / If-None-Match，其中Etag / If-None-Match的优先级比Last-Modified / If-Modified-Since高。协商缓存主要有以下两种情况：

* 1. 协商缓存生效，返回304
  2. 协商缓存失效，返回200和请求结果结果

传送门 ☞ [# 彻底理解浏览器的缓存机制](https://juejin.cn/post/6992843117963509791" \o "https://juejin.cn/post/6992843117963509791" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### 说下进程、线程和协程

**进程**是一个具有一定独立功能的程序在一个数据集上的一次动态执行的过程，是操作系统进行资源分配和调度的一个独立单位，是应用程序运行的载体。进程是一种抽象的概念，从来没有统一的标准定义。

**线程**是程序执行中一个单一的顺序控制流程，是程序执行流的最小单元，是处理器调度和分派的基本单位。一个进程可以有一个或多个线程，各个线程之间共享程序的内存空间(也就是所在进程的内存空间)。一个标准的线程由线程ID、当前指令指针(PC)、寄存器和堆栈组成。而进程由内存空间(代码、数据、进程空间、打开的文件)和一个或多个线程组成。

**协程**，英文Coroutines，是一种基于线程之上，但又比线程更加轻量级的存在，这种由程序员自己写程序来管理的轻量级线程叫做『用户空间线程』，具有对内核来说不可见的特性。

**进程和线程的区别与联系**

【区别】：  
调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位；  
并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行；  
拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源。  
系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。但是进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个进程死掉就等于所有的线程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。

【联系】： 一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程；  
资源分配给进程，同一进程的所有线程共享该进程的所有资源；  
处理机分给线程，即真正在处理机上运行的是线程；  
线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

传送门 ☞ [# 一文搞懂进程、线程、协程及JS协程的发展](https://juejin.cn/post/7005465381791875109" \o "https://juejin.cn/post/7005465381791875109" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank) [☞了解更多](https://link.juejin.cn?target=http://www.360doc.com/content/20/0417/14/32196507_906628857.shtml" \o "http://www.360doc.com/content/20/0417/14/32196507_906628857.shtml" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

关于浏览器传送门 ☞[# 深入了解现代 Web 浏览器](https://juejin.cn/post/6993095345576083486" \o "https://juejin.cn/post/6993095345576083486" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

## HTML && CSS

### HTML5 新特性、语义化

**概念**：

HTML5的语义化指的是合理正确的使用语义化的标签来创建页面结构。【正确的标签做正确的事】

**语义化标签**：

header nav main article section aside footer

**语义化的优点**:

* + 在没CSS样式的情况下，页面整体也会呈现很好的结构效果
  + 代码结构清晰，易于阅读，
  + 利于开发和维护 方便其他设备解析（如屏幕阅读器）根据语义渲染网页。
  + 有利于搜索引擎优化（SEO），搜索引擎爬虫会根据不同的标签来赋予不同的权重

### CSS 选择器及优先级

**选择器**

* id选择器(#myid)
* 类选择器(.myclass)
* 属性选择器(a[rel="external"])
* 伪类选择器(a:hover, li:nth-child)
* 标签选择器(div, h1,p)
* 相邻选择器（h1 + p）
* 子选择器(ul > li)
* 后代选择器(li a)
* 通配符选择器(\*)

**优先级：**

* !important
* 内联样式（1000）
* ID选择器（0100）
* 类选择器/属性选择器/伪类选择器（0010）
* 元素选择器/伪元素选择器（0001）
* 关系选择器/通配符选择器（0000）

带!important 标记的样式属性优先级最高； 样式表的来源相同时：!important > 行内样式>ID选择器 > 类选择器 > 标签 > 通配符 > 继承 > 浏览器默认属性

### position 属性的值有哪些及其区别

**固定定位 fixed**： 元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。Fixed 定 位使元素的位置与文档流无关，因此不占据空间。 Fixed 定位的元素和其他元素重叠。

**相对定位 relative**： 如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直 或水平位置，让这个元素“相对于”它的起点进行移动。 在使用相对定位时，无论是 否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。

**绝对定位 absolute**： 绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那 么它的位置相对于。absolute 定位使元素的位置与文档流无关，因此不占据空间。 absolute 定位的元素和其他元素重叠。

**粘性定位 sticky**： 元素先按照普通文档流定位，然后相对于该元素在流中的 flow root（BFC）和 containing block（最近的块级祖先元素）定位。而后，元素定位表现为在跨越特定阈值前为相对定 位，之后为固定定位。

**默认定位 Static**： 默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声 明）。 inherit: 规定应该从父元素继承 position 属性的值。

### box-sizing属性

box-sizing 规定两个并排的带边框的框，语法为 box-sizing：content-box/border-box/inherit

**content-box**：宽度和高度分别应用到元素的内容框，在宽度和高度之外绘制元素的内边距和边框。【标准盒子模型】

**border-box**：为元素设定的宽度和高度决定了元素的边框盒。【IE 盒子模型】

**inherit**：继承父元素的 box-sizing 值。

### CSS 盒子模型

CSS 盒模型本质上是一个盒子，它包括：边距，边框，填充和实际内容。CSS 中的盒子模型包括 IE 盒子模型和标准的 W3C 盒子模型。  
在标准的盒子模型中，width 指 content 部分的宽度。  
在 IE 盒子模型中，width 表示 content+padding+border 这三个部分的宽度。

故在计算盒子的宽度时存在差异：

**标准盒模型：** 一个块的总宽度 = width+margin(左右)+padding(左右)+border(左右)

**怪异盒模型：** 一个块的总宽度 = width+margin（左右）（既 width 已经包含了 padding 和 border 值）

### BFC（块级格式上下文）

**BFC的概念**

BFC 是 Block Formatting Context 的缩写，即块级格式化上下文。BFC是CSS布局的一个概念，是一个独立的渲染区域，规定了内部box如何布局， 并且这个区域的子元素不会影响到外面的元素，其中比较重要的布局规则有内部 box 垂直放置，计算 BFC 的高度的时候，浮动元素也参与计算。

**BFC的原理布局规则**

* 内部的Box会在垂直方向，一个接一个地放置
* Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠
* 每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反
* BFC的区域不会与float box重叠
* BFC是一个独立容器，容器里面的子元素不会影响到外面的元素
* 计算BFC的高度时，浮动元素也参与计算高度
* 元素的类型和display属性，决定了这个Box的类型。不同类型的Box会参与不同的Formatting Context。

**如何创建BFC？**

* 根元素，即HTML元素
* float的值不为none
* position为absolute或fixed
* display的值为inline-block、table-cell、table-caption
* overflow的值不为visible

**BFC的使用场景**

* 去除边距重叠现象
* 清除浮动（让父元素的高度包含子浮动元素）
* 避免某元素被浮动元素覆盖
* 避免多列布局由于宽度计算四舍五入而自动换行

### 让一个元素水平垂直居中

**水平居中**

对于 行内元素 : text-align: center;

对于确定宽度的块级元素：

（1）width和margin实现。margin: 0 auto;

（2）绝对定位和margin-left: (父width - 子width）/2, 前提是父元素position: relative

对于宽度未知的块级元素

（1）table标签配合margin左右auto实现水平居中。使用table标签（或直接将块级元素设值为 display:table），再通过给该标签添加左右margin为auto。

（2）inline-block实现水平居中方法。display：inline-block和text-align:center实现水平居中。

（3）绝对定位+transform，translateX可以移动本身元素的50%。

（4）flex布局使用justify-content:center

**垂直居中**

* 1. 利用 line-height 实现居中，这种方法适合纯文字类
  2. 通过设置父容器 相对定位 ，子级设置 绝对定位，标签通过margin实现自适应居中
  3. 弹性布局 flex :父级设置display: flex; 子级设置margin为auto实现自适应居中
  4. 父级设置相对定位，子级设置绝对定位，并且通过位移 transform 实现
  5. table 布局，父级通过转换成表格形式，然后子级设置 vertical-align 实现。（需要注意的是：vertical-align: middle使用的前提条件是内联元素以及display值为table-cell的元素）。

传送门 ☞ [# 图解CSS水平垂直居中常见面试方法](https://juejin.cn/post/7008348524530106381" \o "https://juejin.cn/post/7008348524530106381" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### 隐藏页面中某个元素的方法

1.opacity：0，该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定 一些事件，如click 事件，那么点击该区域，也能触发点击事件的

2.visibility：hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已 经绑定的事件 ，隐藏对应元素，在文档布局中仍保留原来的空间（重绘）

3.display：none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素。 不显示对应的元素，在文档布局中不再分配空间（回流+重绘）

该问题会引出 回流和重绘

### 用CSS实现三角符号

/\*记忆口诀：盒子宽高均为零，三面边框皆透明。 \*/div:after{

position: absolute;

width: 0px;

height: 0px;

content: " ";

border-right: 100px solid transparent;

border-top: 100px solid #ff0;

border-left: 100px solid transparent;

border-bottom: 100px solid transparent;

}复制代码

### 页面布局

#### 1.Flex 布局

布局的传统解决方案，基于盒状模型，依赖 display 属性 + position 属性 + float 属性。它对于那些特殊布局非常不方便，比如，垂直居中就不容易实现。

Flex 是 Flexible Box 的缩写，意为"弹性布局",用来为盒状模型提供最大的灵活性。指定容器 display: flex 即可。 简单的分为容器属性和元素属性。

容器的属性：

* flex-direction：决定主轴的方向（即子 item 的排列方法）flex-direction: row | row-reverse | column | column-reverse;
* flex-wrap：决定换行规则 flex-wrap: nowrap | wrap | wrap-reverse;
* flex-flow： .box { flex-flow: || ; }
* justify-content：对其方式，水平主轴对齐方式
* align-items：对齐方式，竖直轴线方向
* align-content

项目的属性（元素的属性）：

* order 属性：定义项目的排列顺序，顺序越小，排列越靠前，默认为 0
* flex-grow 属性：定义项目的放大比例，即使存在空间，也不会放大
* flex-shrink 属性：定义了项目的缩小比例，当空间不足的情况下会等比例的缩小，如果 定义个 item 的 flow-shrink 为 0，则为不缩小
* flex-basis 属性：定义了在分配多余的空间，项目占据的空间。
* flex：是 flex-grow 和 flex-shrink、flex-basis 的简写，默认值为 0 1 auto。
* align-self：允许单个项目与其他项目不一样的对齐方式，可以覆盖
* align-items，默认属 性为 auto，表示继承父元素的 align-items 比如说，用 flex 实现圣杯布局

#### 2.Rem 布局

首先 Rem 相对于根(html)的 font-size 大小来计算。简单的说它就是一个相对单例 如:font-size:10px;,那么（1rem = 10px）了解计算原理后首先解决怎么在不同设备上设置 html 的 font-size 大小。其实 rem 布局的本质是等比缩放，一般是基于宽度。

**优点**：可以快速适用移动端布局，字体，图片高度

**缺点**：

①目前 ie 不支持，对 pc 页面来讲使用次数不多；  
②数据量大：所有的图片，盒子都需要我们去给一个准确的值；才能保证不同机型的适配；  
③在响应式布局中，必须通过 js 来动态控制根元素 font-size 的大小。也就是说 css 样式和 js 代码有一定的耦合性。且必须将改变 font-size 的代码放在 css 样式之前。

#### 3.百分比布局

通过百分比单位 " % " 来实现响应式的效果。通过百分比单位可以使得浏览器中的组件的宽和高随着浏览器的变化而变化，从而实现响应式的效果。 直观的理解，我们可能会认为子元素的百分比完全相对于直接父元素，height 百分比相 对于 height，width 百分比相对于 width。 padding、border、margin 等等不论是垂直方向还是水平方向，都相对于直接父元素的 width。 除了 border-radius 外，还有比如 translate、background-size 等都是相对于自身的。

**缺点**：

（1）计算困难  
（2）各个属性中如果使用百分比，相对父元素的属性并不是唯一的。造成我们使用百分比单位容易使布局问题变得复杂。

#### 4.浮动布局

浮动布局:当元素浮动以后可以向左或向右移动，直到它的外边缘碰到包含它的框或者另外一个浮动元素的边框为止。元素浮动以后会脱离正常的文档流，所以文档的普通流中的框就变的好像浮动元素不存在一样。

**优点**

这样做的优点就是在图文混排的时候可以很好的使文字环绕在图片周围。另外当元素浮动了起来之后，它有着块级元素的一些性质例如可以设置宽高等，但它与inline-block还是有一些区别的，第一个就是关于横向排序的时候，float可以设置方向而inline-block方向是固定的；还有一个就是inline-block在使用时有时会有空白间隙的问题

**缺点**

最明显的缺点就是浮动元素一旦脱离了文档流，就无法撑起父元素，会造成父级元素高度塌陷。

### 如何使用rem或viewport进行移动端适配

**rem适配原理：**

改变了一个元素在不同设备上占据的css像素的个数

rem适配的优缺点

* 优点：没有破坏完美视口
* 缺点：px值转换rem太过于复杂(下面我们使用less来解决这个问题)

**viewport适配的原理**

viewport适配方案中，每一个元素在不同设备上占据的css像素的个数是一样的。但是css像素和物理像素的比例是不一样的，等比的

viewport适配的优缺点

* 在我们设计图上所量取的大小即为我们可以设置的像素大小，即所量即所设
* 缺点破坏完美视口

### 清除浮动的方式

* 添加额外标签

<div class="parent">

//添加额外标签并且添加clear属性

<div style="clear:both"></div>

//也可以加一个br标签</div>复制代码

* 父级添加overflow属性，或者设置高度
* 建立伪类选择器清除浮动

//在css中添加:after伪元素

.parent:after{

/\* 设置添加子元素的内容是空 \*/

content: '';

/\* 设置添加子元素为块级元素 \*/

display: block;

/\* 设置添加的子元素的高度0 \*/

height: 0;

/\* 设置添加子元素看不见 \*/

visibility: hidden;

/\* 设置clear：both \*/

clear: both;

}复制代码

## JS、TS、ES6

### JS中的8种数据类型及区别

包括值类型(基本对象类型)和引用类型(复杂对象类型)

**基本类型(值类型)：** Number(数字),String(字符串),Boolean(布尔),Symbol(符号),null(空),undefined(未定义)在内存中占据固定大小，保存在栈内存中

**引用类型(复杂数据类型)：** Object(对象)、Function(函数)。其他还有Array(数组)、Date(日期)、RegExp(正则表达式)、特殊的基本包装类型(String、Number、Boolean) 以及单体内置对象(Global、Math)等 引用类型的值是对象 保存在堆内存中，栈内存存储的是对象的变量标识符以及对象在堆内存中的存储地址。

传送门 ☞[# JavaScript 数据类型之 Symbol、BigInt](https://juejin.cn/post/7000754813801775111" \o "https://juejin.cn/post/7000754813801775111" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### JS中的数据类型检测方案

#### 1.typeof

console.log(typeof 1); // numberconsole.log(typeof true); // booleanconsole.log(typeof 'mc'); // stringconsole.log(typeof Symbol) // functionconsole.log(typeof function(){}); // functionconsole.log(typeof console.log()); // functionconsole.log(typeof []); // object console.log(typeof {}); // objectconsole.log(typeof null); // objectconsole.log(typeof undefined); // undefined复制代码

优点：能够快速区分基本数据类型

缺点：不能将Object、Array和Null区分，都返回object

#### 2.instanceof

console.log(1 instanceof Number); // falseconsole.log(true instanceof Boolean); // false console.log('str' instanceof String); // false console.log([] instanceof Array); // trueconsole.log(function(){} instanceof Function); // trueconsole.log({} instanceof Object); // true复制代码

优点：能够区分Array、Object和Function，适合用于判断自定义的类实例对象

缺点：Number，Boolean，String基本数据类型不能判断

#### 3.Object.prototype.toString.call()

var toString = Object.prototype.toString;console.log(toString.call(1)); //[object Number]console.log(toString.call(true)); //[object Boolean]console.log(toString.call('mc')); //[object String]console.log(toString.call([])); //[object Array]console.log(toString.call({})); //[object Object]console.log(toString.call(function(){})); //[object Function]console.log(toString.call(undefined)); //[object Undefined]console.log(toString.call(null)); //[object Null]复制代码

优点：精准判断数据类型

缺点：写法繁琐不容易记，推荐进行封装后使用

### var && let && const

ES6之前创建变量用的是var,之后创建变量用的是let/const

**三者区别**：

1. var定义的变量，没有块的概念，可以跨块访问, 不能跨函数访问。  
   let定义的变量，只能在块作用域里访问，不能跨块访问，也不能跨函数访问。  
   const用来定义常量，使用时必须初始化(即必须赋值)，只能在块作用域里访问，且不能修改。
2. var可以先使用，后声明，因为存在变量提升；let必须先声明后使用。
3. var是允许在相同作用域内重复声明同一个变量的，而let与const不允许这一现象。
4. 在全局上下文中，基于let声明的全局变量和全局对象GO（window）没有任何关系 ;  
   var声明的变量会和GO有映射关系；
5. 会产生暂时性死区：

暂时性死区是浏览器的bug：检测一个未被声明的变量类型时，不会报错，会返回undefined  
如：console.log(typeof a) //undefined  
而：console.log(typeof a)//未声明之前不能使用  
let a

1. let /const/function会把当前所在的大括号(除函数之外)作为一个全新的块级上下文，应用这个机制，在开发项目的时候，遇到循环事件绑定等类似的需求，无需再自己构建闭包来存储，只要基于let的块作用特征即可解决

### JS垃圾回收机制

项目中，如果存在大量不被释放的内存（堆/栈/上下文），页面性能会变得很慢。当某些代码操作不能被合理释放，就会造成内存泄漏。我们尽可能减少使用闭包，因为它会消耗内存。

浏览器垃圾回收机制/内存回收机制:

浏览器的Javascript具有自动垃圾回收机制(GC:Garbage Collecation)，垃圾收集器会定期（周期性）找出那些不在继续使用的变量，然后释放其内存。

**标记清除**:在js中，最常用的垃圾回收机制是标记清除：当变量进入执行环境时，被标记为“进入环境”，当变量离开执行环境时，会被标记为“离开环境”。垃圾回收器会销毁那些带标记的值并回收它们所占用的内存空间。  
**谷歌浏览器**：“查找引用”，浏览器不定时去查找当前内存的引用，如果没有被占用了，浏览器会回收它；如果被占用，就不能回收。  
**IE浏览器**：“引用计数法”，当前内存被占用一次，计数累加1次，移除占用就减1，减到0时，浏览器就回收它。

优化手段：内存优化 ; 手动释放：取消内存的占用即可。

（1）堆内存：fn = null 【null：空指针对象】

（2）栈内存：把上下文中，被外部占用的堆的占用取消即可。

内存泄漏

在 JS 中，常见的内存泄露主要有 4 种,全局变量、闭包、DOM 元素的引用、定时器

### 作用域和作用域链

创建函数的时候，已经声明了当前函数的作用域==>当前创建函数所处的上下文。如果是在全局下创建的函数就是[[scope]]:EC(G)，函数执行的时候，形成一个全新的私有上下文EC(FN)，供字符串代码执行(进栈执行)

定义：简单来说作用域就是变量与函数的可访问范围，由当前环境与上层环境的一系列变量对象组成  
1.全局作用域：代码在程序的任何地方都能被访问，window 对象的内置属性都拥有全局作用域。  
2.函数作用域：在固定的代码片段才能被访问

作用：作用域最大的用处就是隔离变量，不同作用域下同名变量不会有冲突。

**作用域链参考链接**一般情况下，变量到 创建该变量 的函数的作用域中取值。但是如果在当前作用域中没有查到，就会向上级作用域去查，直到查到全局作用域，这么一个查找过程形成的链条就叫做作用域链。

### 闭包的两大作用：保存/保护

**闭包的概念**

函数执行时形成的私有上下文EC(FN)，正常情况下，代码执行完会出栈后释放;但是特殊情况下，如果当前私有上下文中的某个东西被上下文以外的事物占用了，则上下文不会出栈释放，从而形成不销毁的上下文。 函数执行函数执行过程中，会形成一个全新的私有上下文，可能会被释放，可能不会被释放，不论释放与否，他的作用是：

（1）保护：划分一个独立的代码执行区域，在这个区域中有自己私有变量存储的空间，保护自己的私有变量不受外界干扰（操作自己的私有变量和外界没有关系）；

（2）保存：如果当前上下文不被释放【只要上下文中的某个东西被外部占用即可】，则存储的这些私有变量也不会被释放，可以供其下级上下文中调取使用，相当于把一些值保存起来了；

我们把函数执行形成私有上下文，来保护和保存私有变量机制称为闭包。

闭包是指有权访问另一个函数作用域中的变量的函数--《JavaScript高级程序设计》

**稍全面的回答**： 在js中变量的作用域属于函数作用域, 在函数执行完后,作用域就会被清理,内存也会随之被回收,但是由于闭包函数是建立在函数内部的子函数, 由于其可访问上级作用域,即使上级函数执行完, 作用域也不会随之销毁, 这时的子函数(也就是闭包),便拥有了访问上级作用域中变量的权限,即使上级函数执行完后作用域内的值也不会被销毁。

**闭包的特性**：

1、内部函数可以访问定义他们外部函数的参数和变量。(作用域链的向上查找，把外围的作用域中的变量值存储在内存中而不是在函数调用完毕后销毁)设计私有的方法和变量，避免全局变量的污染。

1.1.闭包是密闭的容器，，类似于set、map容器，存储数据的

1.2.闭包是一个对象，存放数据的格式为 key-value 形式

2、函数嵌套函数

3、本质是将函数内部和外部连接起来。优点是可以读取函数内部的变量，让这些变量的值始终保存在内存中，不会在函数被调用之后自动清除

**闭包形成的条件**：

* 1. 函数的嵌套
  2. 内部函数引用外部函数的局部变量，延长外部函数的变量生命周期

**闭包的用途**：

* 1. 模仿块级作用域
  2. 保护外部函数的变量 能够访问函数定义时所在的词法作用域(阻止其被回收)
  3. 封装私有化变量
  4. 创建模块

**闭包应用场景**

闭包的两个场景，闭包的两大作用：保存/保护。 在开发中, 其实我们随处可见闭包的身影, 大部分前端JavaScript 代码都是“事件驱动”的,即一个事件绑定的回调方法; 发送ajax请求成功|失败的回调;setTimeout的延时回调;或者一个函数内部返回另一个匿名函数,这些都是闭包的应用。

**闭包的优点**：延长局部变量的生命周期

**闭包缺点**：会导致函数的变量一直保存在内存中，过多的闭包可能会导致内存泄漏

### JS 中 this 的五种情况

1. 作为普通函数执行时，this指向window。
2. 当函数作为对象的方法被调用时，this就会指向该对象。
3. 构造器调用，this指向返回的这个对象。
4. 箭头函数 箭头函数的this绑定看的是this所在函数定义在哪个对象下，就绑定哪个对象。如果有嵌套的情况，则this绑定到最近的一层对象上。
5. 基于Function.prototype上的 apply 、 call 和 bind 调用模式，这三个方法都可以显示的指定调用函数的 this 指向。apply接收参数的是数组，call接受参数列表，`` bind方法通过传入一个对象，返回一个 this 绑定了传入对象的新函数。这个函数的 this指向除了使用new `时会被改变，其他情况下都不会改变。若为空默认是指向全局对象window。

### 原型 && 原型链

**原型关系：**

* 每个 class都有显示原型 prototype
* 每个实例都有隐式原型 \_ proto\_
* 实例的\_ proto\_指向对应 class 的 prototype

‌ **原型:**  在 JS 中，每当定义一个对象（函数也是对象）时，对象中都会包含一些预定义的属性。其中每个函数对象都有一个prototype 属性，这个属性指向函数的原型对象。

原型链：函数的原型链对象constructor默认指向函数本身，原型对象除了有原型属性外，为了实现继承，还有一个原型链指针\_\_proto\_\_,该指针是指向上一层的原型对象，而上一层的原型对象的结构依然类似。因此可以利用\_\_proto\_\_一直指向Object的原型对象上，而Object原型对象用Object.prototype.\_\_ proto\_\_ = null表示原型链顶端。如此形成了js的原型链继承。同时所有的js对象都有Object的基本防范

**特点:**  JavaScript对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

### new运算符的实现机制

1. 首先创建了一个新的空对象
2. 设置原型，将对象的原型设置为函数的prototype对象。
3. 让函数的this指向这个对象，执行构造函数的代码（为这个新对象添加属性）
4. 判断函数的返回值类型，如果是值类型，返回创建的对象。如果是引用类型，就返回这个引用类型的对象。

### EventLoop 事件循环

JS是单线程的，为了防止一个函数执行时间过长阻塞后面的代码，所以会先将同步代码压入执行栈中，依次执行，将异步代码推入异步队列，异步队列又分为宏任务队列和微任务队列，因为宏任务队列的执行时间较长，所以微任务队列要优先于宏任务队列。微任务队列的代表就是，Promise.then，MutationObserver，宏任务的话就是setImmediate setTimeout setInterval

JS运行的环境。一般为浏览器或者Node。 在浏览器环境中，有JS 引擎线程和渲染线程，且两个线程互斥。 Node环境中，只有JS 线程。 不同环境执行机制有差异，不同任务进入不同Event Queue队列。 当主程结束，先执行准备好微任务，然后再执行准备好的宏任务，一个轮询结束。

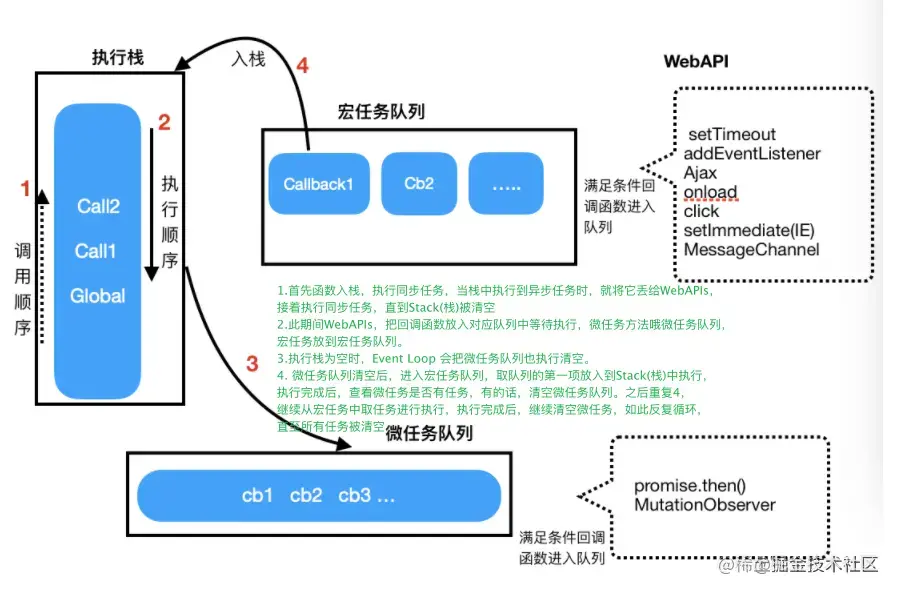
#### ****浏览器中的事件环（Event Loop)****

事件环的运行机制是，先会执行栈中的内容，栈中的内容执行后执行微任务，微任务清空后再执行宏任务，先取出一个宏任务，再去执行微任务，然后在取宏任务清微任务这样不停的循环。

eventLoop 是由JS的宿主环境（浏览器）来实现的；

事件循环可以简单的描述为以下四个步骤:

* 1. 函数入栈，当Stack中执行到异步任务的时候，就将他丢给WebAPIs,接着执行同步任务,直到Stack为空；
  2. 此期间WebAPIs完成这个事件，把回调函数放入队列中等待执行（微任务放到微任务队列，宏任务放到宏任务队列）
  3. 执行栈为空时，Event Loop把微任务队列执行清空；
  4. 微任务队列清空后，进入宏任务队列，取队列的第一项任务放入Stack(栈）中执行，执行完成后，查看微任务队列是否有任务，有的话，清空微任务队列。重复4，继续从宏任务中取任务执行，执行完成之后，继续清空微任务，如此反复循环，直至清空所有的任务。



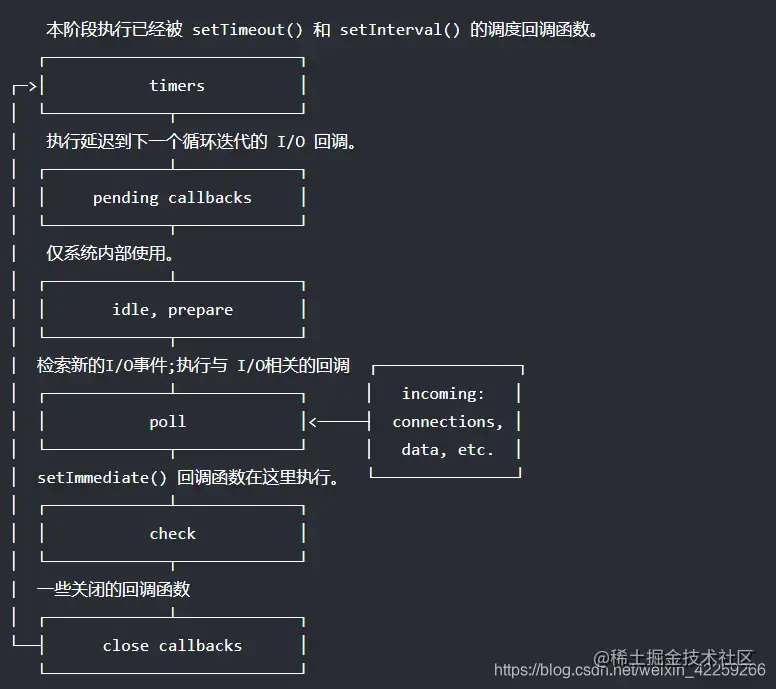
浏览器中的任务源(task):

* + 宏任务(macrotask)：  
    宿主环境提供的，比如浏览器  
    ajax、setTimeout、setInterval、setTmmediate(只兼容ie)、script、requestAnimationFrame、messageChannel、UI渲染、一些浏览器api
  + 微任务(microtask)：  
    语言本身提供的，比如promise.then  
    then、queueMicrotask(基于then)、mutationObserver(浏览器提供)、messageChannel 、mutationObersve

传送门 ☞ [# 宏任务和微任务](https://juejin.cn/post/7001881781125251086" \o "https://juejin.cn/post/7001881781125251086" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

#### ****Node 环境中的事件环（Event Loop)****

Node是基于V8引擎的运行在服务端的JavaScript运行环境，在处理高并发、I/O密集(文件操作、网络操作、数据库操作等)场景有明显的优势。虽然用到也是V8引擎，但由于服务目的和环境不同，导致了它的API与原生JS有些区别，其Event Loop还要处理一些I/O，比如新的网络连接等，所以Node的Event Loop(事件环机制)与浏览器的是不太一样。

执行顺序如下：

* timers: 计时器，执行setTimeout和setInterval的回调
* pending callbacks: 执行延迟到下一个循环迭代的 I/O 回调
* idle, prepare: 队列的移动，仅系统内部使用
* poll轮询: 检索新的 I/O 事件;执行与 I/O 相关的回调。事实上除了其他几个阶段处理的事情，其他几乎所有的异步都在这个阶段处理。
* check: 执行setImmediate回调，setImmediate在这里执行
* close callbacks: 执行close事件的callback，一些关闭的回调函数，如：socket.on('close', ...)

### setTimeout、Promise、Async/Await 的区别

setTimeout

settimeout的回调函数放到宏任务队列里，等到执行栈清空以后执行。

Promise

Promise本身是**同步的立即执行函数**， 当在executor中执行resolve或者reject的时候, 此时是异步操作， 会先执行then/catch等，当主栈完成后，才会去调用resolve/reject中存放的方法执行。

console.log('script start')let promise1 = new Promise(function (resolve) {

console.log('promise1')

resolve()

console.log('promise1 end')

}).then(function () {

console.log('promise2')

})setTimeout(function(){

console.log('settimeout')

})console.log('script end')// 输出顺序: script start->promise1->promise1 end->script end->promise2->settimeout复制代码

async/await

async 函数返回一个 Promise 对象，当函数执行的时候，一旦遇到 await 就会先返回，等到触发的异步操作完成，再执行函数体内后面的语句。可以理解为，是让出了线程，跳出了 async 函数体。

async function async1(){

console.log('async1 start');

await async2();

console.log('async1 end')

}async function async2(){

console.log('async2')

}

console.log('script start');

async1();console.log('script end')

// 输出顺序：script start->async1 start->async2->script end->async1 end复制代码

传送门 ☞ [# JavaScript Promise 专题](https://juejin.cn/post/6999651011304357925" \o "https://juejin.cn/post/6999651011304357925" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### Async/Await 如何通过同步的方式实现异步

Async/Await就是一个**自执行**的generate函数。利用generate函数的特性把异步的代码写成“同步”的形式,第一个请求的返回值作为后面一个请求的参数,其中每一个参数都是一个promise对象.

### 介绍节流防抖原理、区别以及应用

节流：事件触发后，规定时间内，事件处理函数不能再次被调用。也就是说在规定的时间内，函数只能被调用一次，且是最先被触发调用的那次。

防抖：多次触发事件，事件处理函数只能执行一次，并且是在触发操作结束时执行。也就是说，当一个事件被触发准备执行事件函数前，会等待一定的时间（这时间是码农自己去定义的，比如 1 秒），如果没有再次被触发，那么就执行，如果被触发了，那就本次作废，重新从新触发的时间开始计算，并再次等待 1 秒，直到能最终执行！

使用场景：  
节流：滚动加载更多、搜索框搜的索联想功能、高频点击、表单重复提交……  
防抖：搜索框搜索输入，并在输入完以后自动搜索、手机号，邮箱验证输入检测、窗口大小 resize 变化后，再重新渲染。

/\*\*

\* 节流函数 一个函数执行一次后，只有大于设定的执行周期才会执行第二次。有个需要频繁触发的函数，出于优化性能的角度，在规定时间内，只让函数触发的第一次生效，后面的不生效。

\* @param fn要被节流的函数

\* @param delay规定的时间

\*/function throttle(fn, delay) {

//记录上一次函数触发的时间

var lastTime = 0;

return function(){

//记录当前函数触发的时间

var nowTime = Date.now();

if(nowTime - lastTime > delay){

//修正this指向问题

fn.call(this);

//同步执行结束时间

lastTime = nowTime;

}

}

}

document.onscroll = throttle(function () {

console.log('scllor事件被触发了' + Date.now());

}, 200);

/\*\*

\* 防抖函数 一个需要频繁触发的函数，在规定时间内，只让最后一次生效，前面的不生效

\* @param fn要被节流的函数

\* @param delay规定的时间

\*/function debounce(fn, delay) {

//记录上一次的延时器

var timer = null;

return function () {

//清除上一次的演示器

clearTimeout(timer);

//重新设置新的延时器

timer = setTimeout(function(){

//修正this指向问题

fn.apply(this);

}, delay);

}

}document.getElementById('btn').onclick = debounce(function () {

console.log('按钮被点击了' + Date.now());

}, 1000);复制代码

### 简述MVVM

**什么是MVVM？**

视图模型双向绑定，是Model-View-ViewModel的缩写，也就是把MVC中的Controller演变成ViewModel。Model层代表数据模型，View代表UI组件，ViewModel是View和Model层的桥梁，数据会绑定到viewModel层并自动将数据渲染到页面中，视图变化的时候会通知viewModel层更新数据。以前是操作DOM结构更新视图，现在是数据驱动视图。

**MVVM的优点：**

1.低耦合。视图（View）可以独立于Model变化和修改，一个Model可以绑定到不同的View上，当View变化的时候Model可以不变化，当Model变化的时候View也可以不变；  
2.可重用性。你可以把一些视图逻辑放在一个Model里面，让很多View重用这段视图逻辑。  
3.独立开发。开发人员可以专注于业务逻辑和数据的开发(ViewModel)，设计人员可以专注于页面设计。  
4.可测试。

### Vue底层实现原理

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和getter，在数据变动时发布消息给订阅者，触发相应的监听回调  
Vue是一个典型的MVVM框架，模型（Model）只是普通的javascript对象，修改它则试图（View）会自动更新。这种设计让状态管理变得非常简单而直观

**Observer（数据监听器）** : Observer的核心是通过Object.defineProprtty()来监听数据的变动，这个函数内部可以定义setter和getter，每当数据发生变化，就会触发setter。这时候Observer就要通知订阅者，订阅者就是Watcher

**Watcher（订阅者）** : Watcher订阅者作为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

**Compile（指令解析器）** : Compile主要做的事情是解析模板指令，将模板中变量替换成数据，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，添加鉴定数据的订阅者，一旦数据有变动，收到通知，更新试图

### 谈谈对vue生命周期的理解？

每个Vue实例在创建时都会经过一系列的初始化过程，vue的生命周期钩子，就是说在达到某一阶段或条件时去触发的函数，目的就是为了完成一些动作或者事件

* create阶段：vue实例被创建  
  beforeCreate: 创建前，此时data和methods中的数据都还没有初始化  
  created： 创建完毕，data中有值，未挂载
* mount阶段： vue实例被挂载到真实DOM节点  
  beforeMount：可以发起服务端请求，去数据  
  mounted: 此时可以操作DOM
* update阶段：当vue实例里面的data数据变化时，触发组件的重新渲染  
  beforeUpdate :更新前  
  updated：更新后
* destroy阶段：vue实例被销毁  
  beforeDestroy：实例被销毁前，此时可以手动销毁一些方法  
  destroyed:销毁后

#### 组件生命周期

生命周期（父子组件） 父组件beforeCreate --> 父组件created --> 父组件beforeMount --> 子组件beforeCreate --> 子组件created --> 子组件beforeMount --> 子组件 mounted --> 父组件mounted -->父组件beforeUpdate -->子组件beforeDestroy--> 子组件destroyed --> 父组件updated

**加载渲染过程** 父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

**挂载阶段** 父created->子created->子mounted->父mounted

**父组件更新阶段** 父beforeUpdate->父updated

**子组件更新阶段** 父beforeUpdate->子beforeUpdate->子updated->父updated

**销毁阶段** 父beforeDestroy->子beforeDestroy->子destroyed->父destroyed

### computed与watch

通俗来讲，既能用 computed 实现又可以用 watch 监听来实现的功能，推荐用 computed， 重点在于 computed 的缓存功能 computed 计算属性是用来声明式的描述一个值依赖了其它的值，当所依赖的值或者变量 改变时，计算属性也会跟着改变； watch 监听的是已经在 data 中定义的变量，当该变量变化时，会触发 watch 中的方法。

**watch 属性监听** 是一个对象，键是需要观察的属性，值是对应回调函数，主要用来监听某些特定数据的变化，从而进行某些具体的业务逻辑操作,监听属性的变化，需要在数据变化时执行异步或开销较大的操作时使用

**computed 计算属性** 属性的结果会被缓存，当computed中的函数所依赖的属性没有发生改变的时候，那么调用当前函数的时候结果会从缓存中读取。除非依赖的响应式属性变化时才会重新计算，主要当做属性来使用 computed中的函数必须用return返回最终的结果 computed更高效，优先使用。data 不改变，computed 不更新。

**使用场景** computed：当一个属性受多个属性影响的时候使用，例：购物车商品结算功能 watch：当一条数据影响多条数据的时候使用，例：搜索数据

### 组件中的data为什么是一个函数？

1.一个组件被复用多次的话，也就会创建多个实例。本质上，这些实例用的都是同一个构造函数。 2.如果data是对象的话，对象属于引用类型，会影响到所有的实例。所以为了保证组件不同的实例之间data不冲突，data必须是一个函数。

### 为什么v-for和v-if不建议用在一起

1.当 v-for 和 v-if 处于同一个节点时，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。如果要遍历的数组很大，而真正要展示的数据很少时，这将造成很大的性能浪费  
2.这种场景建议使用 computed，先对数据进行过滤

注意：3.x 版本中 v-if 总是优先于 v-for 生效。由于语法上存在歧义，建议避免在同一元素上同时使用两者。比起在模板层面管理相关逻辑，更好的办法是通过创建计算属性筛选出列表，并以此创建可见元素。

解惑传送门 ☞ [# v-if 与 v-for 的优先级对比非兼容](https://link.juejin.cn?target=https://v3.cn.vuejs.org/guide/migration/v-if-v-for.html%23%E6%A6%82%E8%A7%88" \o "https://v3.cn.vuejs.org/guide/migration/v-if-v-for.html#%E6%A6%82%E8%A7%88" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### React/Vue 项目中 key 的作用

key的作用是为了在diff算法执行时更快的找到对应的节点，提高diff速度，更高效的更新虚拟DOM;

vue和react都是采用diff算法来对比新旧虚拟节点，从而更新节点。在vue的diff函数中，会根据新节点的key去对比旧节点数组中的key，从而找到相应旧节点。如果没找到就认为是一个新增节点。而如果没有key，那么就会采用遍历查找的方式去找到对应的旧节点。一种一个map映射，另一种是遍历查找。相比而言。map映射的速度更快。

为了在数据变化时强制更新组件，以避免“就地复用”带来的副作用。

当 Vue.js 用 v-for 更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。重复的key会造成渲染错误。

### vue组件的通信方式

props/$emit 父子组件通信

父->子props，子->父 $on、$emit 获取父子组件实例 parent、children Ref 获取实例的方式调用组件的属性或者方法 父->子孙 Provide、inject 官方不推荐使用，但是写组件库时很常用

$emit/$on 自定义事件 兄弟组件通信

Event Bus 实现跨组件通信 Vue.prototype.$bus = new Vue() 自定义事件

vuex 跨级组件通信

Vuex、$attrs、$listeners Provide、inject

### nextTick的实现

1. nextTick是Vue提供的一个全局API,是在下次DOM更新循环结束之后执行延迟回调，在修改数据之后使用$nextTick，则可以在回调中获取更新后的DOM；
2. Vue在更新DOM时是异步执行的。只要侦听到数据变化，Vue将开启1个队列，并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，只会被推入到队列中-次。这种在缓冲时去除重复数据对于避免不必要的计算和DOM操作是非常重要的。nextTick方法会在队列中加入一个回调函数，确保该函数在前面的dom操作完成后才调用；
3. 比如，我在干什么的时候就会使用nextTick，传一个回调函数进去，在里面执行dom操作即可；
4. 我也有简单了解nextTick实现，它会在callbacks里面加入我们传入的函数，然后用timerFunc异步方式调用它们，首选的异步方式会是Promise。这让我明白了为什么可以在nextTick中看到dom操作结果。

### nextTick的实现原理是什么？

在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后立即使用 nextTick 来获取更新后的 DOM。 nextTick主要使用了宏任务和微任务。 根据执行环境分别尝试采用Promise、MutationObserver、setImmediate，如果以上都不行则采用setTimeout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。

### 使用过插槽么？用的是具名插槽还是匿名插槽或作用域插槽

vue中的插槽是一个非常好用的东西slot说白了就是一个占位的 在vue当中插槽包含三种一种是默认插槽（匿名）一种是具名插槽还有一种就是作用域插槽 匿名插槽就是没有名字的只要默认的都填到这里具名插槽指的是具有名字的

### keep-alive的实现

作用：实现组件缓存，保持这些组件的状态，以避免反复渲染导致的性能问题。 需要缓存组件 频繁切换，不需要重复渲染

场景：tabs标签页 后台导航，vue性能优化

原理：Vue.js内部将DOM节点抽象成了一个个的VNode节点，keep-alive组件的缓存也是基于VNode节点的而不是直接存储DOM结构。它将满足条件（pruneCache与pruneCache）的组件在cache对象中缓存起来，在需要重新渲染的时候再将vnode节点从cache对象中取出并渲染。

### mixin

mixin 项目变得复杂的时候，多个组件间有重复的逻辑就会用到mixin  
多个组件有相同的逻辑，抽离出来  
mixin并不是完美的解决方案，会有一些问题  
vue3提出的Composition API旨在解决这些问题【追求完美是要消耗一定的成本的，如开发成本】  
场景：PC端新闻列表和详情页一样的右侧栏目，可以使用mixin进行混合  
劣势：1.变量来源不明确，不利于阅读  
2.多mixin可能会造成命名冲突 3.mixin和组件可能出现多对多的关系，使得项目复杂度变高

### Vuex的理解及使用场景

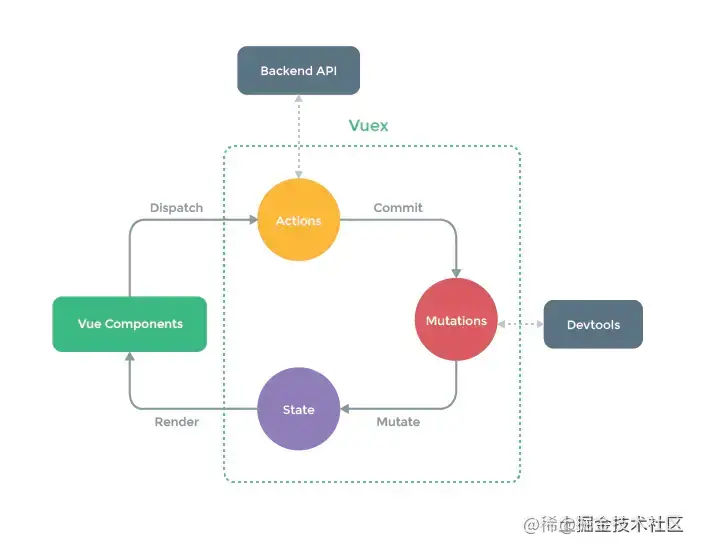
Vuex 是一个专为 Vue 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。

1. Vuex 的状态存储是响应式的；当 Vue 组件从 store 中读取状态的时候，

若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新 2. 改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation， 这样使得我们可以方便地跟踪每一个状态的变化 Vuex主要包括以下几个核心模块：

1. State：定义了应用的状态数据
2. Getter：在 store 中定义“getter”（可以认为是 store 的计算属性），

就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来， 且只有当它的依赖值发生了改变才会被重新计算 3. Mutation：是唯一更改 store 中状态的方法，且必须是同步函数 4. Action：用于提交 mutation，而不是直接变更状态，可以包含任意异步操作 5. Module：允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中



### hooks用过吗？聊聊react中class组件和函数组件的区别

类组件是使用ES6 的 class 来定义的组件。 函数组件是接收一个单一的 props 对象并返回一个React元素。

关于React的两套API（类（class）API 和基于函数的钩子（hooks） API）。官方推荐使用钩子（函数），而不是类。因为钩子更简洁，代码量少，用起来比较"轻"，而类比较"重"。而且，钩子是函数，更符合 React 函数式的本质。

函数一般来说，只应该做一件事，就是返回一个值。 如果你有多个操作，每个操作应该写成一个单独的函数。而且，数据的状态应该与操作方法分离。根据函数这种理念，React 的函数组件只应该做一件事情：返回组件的 HTML 代码，而没有其他的功能。函数的返回结果只依赖于它的参数。不改变函数体外部数据、函数执行过程里面没有副作用。

类（class）是数据和逻辑的封装。 也就是说，组件的状态和操作方法是封装在一起的。如果选择了类的写法，就应该把相关的数据和操作，都写在同一个 class 里面。

**类组件的缺点** :

大型组件很难拆分和重构，也很难测试。  
业务逻辑分散在组件的各个方法之中，导致重复逻辑或关联逻辑。  
组件类引入了复杂的编程模式，比如 render props 和高阶组件。  
难以理解的 class，理解 JavaScript 中 this 的工作方式。

**区别**：

函数组件的性能比类组件的性能要高，因为类组件使用的时候要实例化，而函数组件直接执行函数取返回结果即可。

1.状态的有无  
hooks出现之前，函数组件没有实例，没有生命周期，没有state，没有this，所以我们称函数组件为无状态组件。 hooks出现之前，react中的函数组件通常只考虑负责UI的渲染，没有自身的状态没有业务逻辑代码，是一个纯函数。它的输出只由参数props决定，不受其他任何因素影响。

2.调用方式的不同  
函数组件重新渲染，将重新调用组件方法返回新的react元素。类组件重新渲染将new一个新的组件实例，然后调用render类方法返回react元素，这也说明为什么类组件中this是可变的。

3.因为调用方式不同，在函数组件使用中会出现问题  
在操作中改变状态值，类组件可以获取最新的状态值，而函数组件则会按照顺序返回状态值

**React Hooks（钩子的作用）**

Hook 是 React 16.8 的新增特性。它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性。

React Hooks的几个常用钩子:

1. useState() //状态钩子
2. useContext() //共享状态钩子
3. useReducer() //action 钩子
4. useEffect() //副作用钩子

还有几个不常见的大概的说下，后续会专门写篇文章描述下

1.useCallback 记忆函数 一般把**函数式组件理解为class组件render函数的语法糖**，所以每次重新渲染的时候，函数式组件内部所有的代码都会重新执行一遍。而有了 useCallback 就不一样了，你可以通过 useCallback 获得一个记忆后的函数。

function App() {

const memoizedHandleClick = useCallback(() => {

console.log('Click happened')

}, []); // 空数组代表无论什么情况下该函数都不会发生改变

return <SomeComponent onClick={memoizedHandleClick}>Click Me</SomeComponent>;

}复制代码

第二个参数传入一个数组，数组中的每一项一旦值或者引用发生改变，useCallback 就会重新返回一个新的记忆函数提供给后面进行渲染。

2.useMemo 记忆组件 useCallback 的功能完全可以由 useMemo 所取代，如果你想通过使用 useMemo 返回一个记忆函数也是完全可以的。 唯一的区别是：**useCallback 不会执行第一个参数函数，而是将它返回给你，而 useMemo 会执行第一个函数并且将函数执行结果返回给你**。  
所以 useCallback 常用记忆事件函数，生成记忆后的事件函数并传递给子组件使用。而 useMemo 更适合经过函数计算得到一个确定的值，比如记忆组件。

3.useRef 保存引用值

useRef 跟 createRef 类似，都可以用来生成对 DOM 对象的引用。useRef 返回的值传递给组件或者 DOM 的 ref 属性，就可以通过 ref.current 值**访问组件或真实的 DOM 节点，重点是组件也是可以访问到的**，从而可以对 DOM 进行一些操作，比如监听事件等等。

4.useImperativeHandle 穿透 Ref

通过 useImperativeHandle 用于让父组件获取子组件内的索引

5.useLayoutEffect 同步执行副作用

大部分情况下，使用 useEffect 就可以帮我们处理组件的副作用，但是如果想要同步调用一些副作用，比如对 DOM 的操作，就需要使用 useLayoutEffect，useLayoutEffect 中的副作用会在 DOM 更新之后同步执行。

**useEffect和useLayoutEffect有什么区别**：简单来说就是调用时机不同，useLayoutEffect和原来componentDidMount&componentDidUpdate一致，在react完成DOM更新后马上同步调用的代码，会阻塞页面渲染。而useEffect是会在整个页面渲染完才会调用的代码。官方建议优先使用useEffect

### React 组件通信方式

react组件间通信常见的几种情况:

* 1. 父组件向子组件通信
  2. 子组件向父组件通信
  3. 跨级组件通信
  4. 非嵌套关系的组件通信

#### 1）父组件向子组件通信

父组件通过 props 向子组件传递需要的信息。父传子是在父组件中直接绑定一个正常的属性，这个属性就是指具体的值，在子组件中，用props就可以获取到这个值

// 子组件: Childconst Child = props =>{

return <p>{props.name}</p>

}

// 父组件 Parentconst Parent = ()=>{

return <Child name="京程一灯"></Child>

}复制代码

#### 2）子组件向父组件通信

props+回调的方式，使用公共组件进行状态提升。子传父是先在父组件上绑定属性设置为一个函数，当子组件需要给父组件传值的时候，则通过props调用该函数将参数传入到该函数当中，此时就可以在父组件中的函数中接收到该参数了，这个参数则为子组件传过来的值

// 子组件: Childconst Child = props =>{

const cb = msg =>{

return ()=>{

props.callback(msg)

}

}

return (

<button onClick={cb("京程一灯欢迎你!")}>京程一灯欢迎你</button>

)

}

// 父组件 Parentclass Parent extends Component {

callback(msg){

console.log(msg)

}

render(){

return <Child callback={this.callback.bind(this)}></Child>

}

}复制代码

#### 3）跨级组件通信

即父组件向子组件的子组件通信，向更深层子组件通信。

* 使用props，利用中间组件层层传递,但是如果父组件结构较深，那么中间每一层组件都要去传递props，增加了复杂度，并且这些props并不是中间组件自己需要的。
* 使用context，context相当于一个大容器，我们可以把要通信的内容放在这个容器中，这样不管嵌套多深，都可以随意取用，对于跨越多层的全局数据可以使用context实现。

// context方式实现跨级组件通信 // Context 设计目的是为了共享那些对于一个组件树而言是“全局”的数据

const BatteryContext = createContext();

// 子组件的子组件 class GrandChild extends Component {

render(){

return (

<BatteryContext.Consumer>

{

color => <h1 style={{"color":color}}>我是红色的:{color}</h1>

}

</BatteryContext.Consumer>

)

}

}

// 子组件const Child = () =>{

return (

<GrandChild/>

)

}// 父组件class Parent extends Component {

state = {

color:"red"

}

render(){

const {color} = this.state

return (

<BatteryContext.Provider value={color}>

<Child></Child>

</BatteryContext.Provider>

)

}

}复制代码

#### 4）非嵌套关系的组件通信

即没有任何包含关系的组件，包括兄弟组件以及不在同一个父级中的非兄弟组件。

* 1. 可以使用自定义事件通信（发布订阅模式），使用pubsub-js
  2. 可以通过redux等进行全局状态管理
  3. 如果是兄弟组件通信，可以找到这两个兄弟节点共同的父节点, 结合父子间通信方式进行通信。
  4. 也可以new一个 Vue 的 EventBus,进行事件监听，一边执行监听，一边执行新增 VUE的eventBus 就是发布订阅模式，是可以在React中使用的;

### setState 既存在异步情况也存在同步情况

1.异步情况 在React事件当中是异步操作

2.同步情况 如果是在setTimeout事件或者自定义的dom事件中，都是同步的

//setTimeout事件import React,{ Component } from "react";class Count extends Component{

constructor(props){

super(props);

this.state = {

count:0

}

}

render(){

return (

<>

<p>count:{this.state.count}</p>

<button onClick={this.btnAction}>增加</button>

</>

)

}

btnAction = ()=>{

//不能直接修改state，需要通过setState进行修改

//同步

setTimeout(()=>{

this.setState({

count: this.state.count + 1

});

console.log(this.state.count);

})

}

}

export default Count;复制代码

//自定义dom事件import React,{ Component } from "react";class Count extends Component{

constructor(props){

super(props);

this.state = {

count:0

}

}

render(){

return (

<>

<p>count:{this.state.count}</p>

<button id="btn">绑定点击事件</button>

</>

)

}

componentDidMount(){

//自定义dom事件，也是同步修改

document.querySelector('#btn').addEventListener('click',()=>{

this.setState({

count: this.state.count + 1

});

console.log(this.state.count);

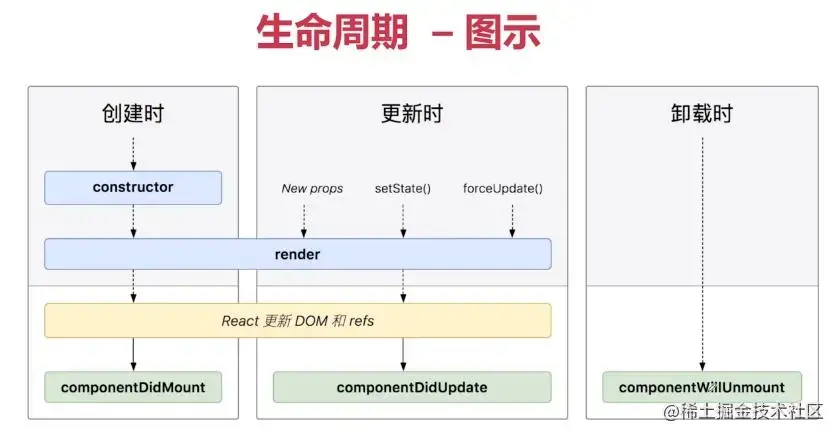
});

}

}

export default Count;复制代码

### 生命周期



安装

当组件的实例被创建并插入到 DOM 中时，这些方法按以下顺序调用：

constructor()static getDerivedStateFromProps()render()componentDidMount()

更新中

更新可能由道具或状态的更改引起。当重新渲染组件时，这些方法按以下顺序调用：

static getDerivedStateFromProps()shouldComponentUpdate()render()getSnapshotBeforeUpdate()componentDidUpdate()

卸载

当组件从 DOM 中移除时调用此方法：

componentWillUnmount()复制代码

### 说一下 react-fiber

#### 1）背景

react-fiber 产生的根本原因，是大量的同步计算任务阻塞了浏览器的 UI 渲染。默认情况下，JS 运算、页面布局和页面绘制都是运行在浏览器的主线程当中，他们之间是互斥的关系。如果 JS 运算持续占用主线程，页面就没法得到及时的更新。当我们调用setState更新页面的时候，React 会遍历应用的所有节点，计算出差异，然后再更新 UI。如果页面元素很多，整个过程占用的时机就可能超过 16 毫秒，就容易出现掉帧的现象。

#### 2）实现原理

react内部运转分三层：

* + Virtual DOM 层，描述页面长什么样。
  + Reconciler 层，负责调用组件生命周期方法，进行 Diff 运算等。
  + Renderer 层，根据不同的平台，渲染出相应的页面，比较常见的是 ReactDOM 和 ReactNative。

Fiber 其实指的是一种数据结构，它可以用一个纯 JS 对象来表示：

const fiber = {

stateNode, // 节点实例

child, // 子节点

sibling, // 兄弟节点

return, // 父节点

}复制代码

为了实现不卡顿，就需要有一个调度器 (Scheduler) 来进行任务分配。优先级高的任务（如键盘输入）可以打断优先级低的任务（如Diff）的执行，从而更快的生效。任务的优先级有六种：

* + synchronous，与之前的Stack Reconciler操作一样，同步执行
  + task，在next tick之前执行
  + animation，下一帧之前执行
  + high，在不久的将来立即执行
  + low，稍微延迟执行也没关系
  + offscreen，下一次render时或scroll时才执行

Fiber Reconciler（react ）执行过程分为2个阶段：

* + 阶段一，生成 Fiber 树，得出需要更新的节点信息。这一步是一个渐进的过程，可以被打断。阶段一可被打断的特性，让优先级更高的任务先执行，从框架层面大大降低了页面掉帧的概率。
  + 阶段二，将需要更新的节点一次过批量更新，这个过程不能被打断。

Fiber树：React 在 render 第一次渲染时，会通过 React.createElement 创建一颗 Element 树，可以称之为 Virtual DOM Tree，由于要记录上下文信息，加入了 Fiber，每一个 Element 会对应一个 Fiber Node，将 Fiber Node 链接起来的结构成为 Fiber Tree。Fiber Tree 一个重要的特点是链表结构，将递归遍历编程循环遍历，然后配合 requestIdleCallback API, 实现任务拆分、中断与恢复。

从Stack Reconciler到Fiber Reconciler，源码层面其实就是干了一件递归改循环的事情

传送门 ☞[# 深入了解 Fiber](https://juejin.cn/post/7002250258826657799" \o "https://juejin.cn/post/7002250258826657799" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### Portals

Portals 提供了一种一流的方式来将子组件渲染到存在于父组件的 DOM 层次结构之外的 DOM 节点中。结构不受外界的控制的情况下就可以使用portals进行创建

### 何时要使用异步组件？如和使用异步组件

* 加载大组件的时候
* 路由异步加载的时候

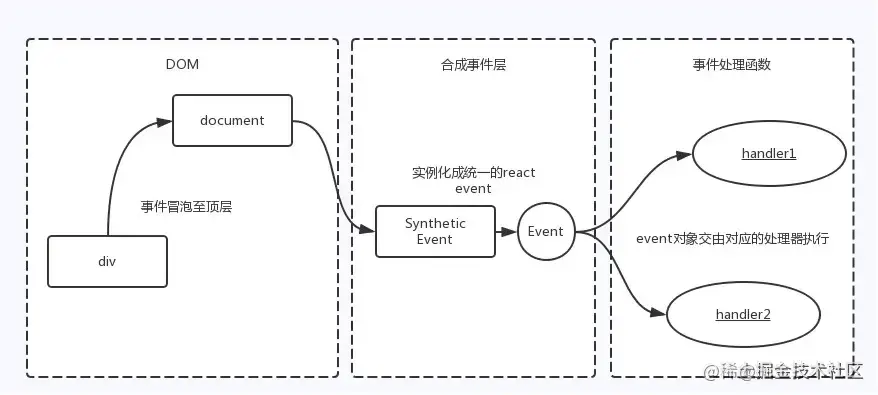
react 中要配合 Suspense 使用

// 异步懒加载const Box = lazy(()=>import('./components/Box'));// 使用组件的时候要用suspense进行包裹<Suspense fallback={<div>loading...</div>}>

{show && <Box/>}</Suspense>复制代码

### React 事件绑定原理

React并不是将click事件绑在该div的真实DOM上，而是在document处监听所有支持的事件，当事件发生并冒泡至document处时，React将事件内容封装并交由真正的处理函数运行。这样的方式不仅减少了内存消耗，还能在组件挂载销毁时统一订阅和移除事件。  
另外冒泡到 document 上的事件也不是原生浏览器事件，而是 React 自己实现的合成事件（SyntheticEvent）。因此我们如果不想要事件冒泡的话，调用 event.stopPropagation 是无效的，而应该调用 event.preventDefault。



## webpack

### webpack 做过哪些优化，开发效率方面、打包策略方面等等

**1）优化 Webpack 的构建速度**

* 使用高版本的 Webpack （使用webpack4）
* 多线程/多实例构建：HappyPack(不维护了)、thread-loader
* 缩小打包作用域：
  + exclude/include (确定 loader 规则范围)
  + resolve.modules 指明第三方模块的绝对路径 (减少不必要的查找)
  + resolve.extensions 尽可能减少后缀尝试的可能性
  + noParse 对完全不需要解析的库进行忽略 (不去解析但仍会打包到 bundle 中，注意被忽略掉的文件里不应该包含 import、require、define 等模块化语句)
  + IgnorePlugin (完全排除模块)
  + 合理使用alias
* 充分利用缓存提升二次构建速度：
  + babel-loader 开启缓存
  + terser-webpack-plugin 开启缓存
  + 使用 cache-loader 或者 hard-source-webpack-plugin  
    注意：thread-loader 和 cache-loader 兩個要一起使用的話，請先放 cache-loader 接著是 thread-loader 最後才是 heavy-loader
* DLL：
  + 使用 DllPlugin 进行分包，使用 DllReferencePlugin(索引链接) 对 manifest.json 引用，让一些基本不会改动的代码先打包成静态资源，避免反复编译浪费时间。

2）使用webpack4-优化原因

* (a)V8带来的优化（for of替代forEach、Map和Set替代Object、includes替代indexOf）
* (b)默认使用更快的md4 hash算法
* (c)webpacks AST可以直接从loader传递给AST，减少解析时间
* (d)使用字符串方法替代正则表达式

①noParse

* 不去解析某个库内部的依赖关系
* 比如jquery 这个库是独立的， 则不去解析这个库内部依赖的其他的东西
* 在独立库的时候可以使用

module.exports = {

module: {

noParse: /jquery/,

rules:[]

}

}复制代码

②IgnorePlugin

* 忽略掉某些内容 不去解析依赖库内部引用的某些内容
* 从moment中引用 ./locol 则忽略掉
* 如果要用local的话 则必须在项目中必须手动引入

import 'moment/locale/zh-cn'module.exports = {

plugins: [

new Webpack.IgnorePlugin(/./local/, /moment/),

]

}复制代码

③dillPlugin

* 不会多次打包， 优化打包时间
* 先把依赖的不变的库打包
* 生成 manifest.json文件
* 然后在webpack.config中引入
* webpack.DllPlugin Webpack.DllReferencePlugin

④happypack -> thread-loader

* 大项目的时候开启多线程打包
* 影响前端发布速度的有两个方面，一个是构建，一个就是压缩，把这两个东西优化起来，可以减少很多发布的时间。

⑤thread-loader  
thread-loader 会将您的 loader 放置在一个 worker 池里面运行，以达到多线程构建。  
把这个 loader 放置在其他 loader 之前（如下图 example 的位置）， 放置在这个 loader 之后的 loader 就会在一个单独的 worker 池(worker pool)中运行。

// webpack.config.jsmodule.exports = {

module: {

rules: [

{

test: /.js$/,

include: path.resolve("src"),

use: [

"thread-loader",

// 你的高开销的loader放置在此 (e.g babel-loader)

]

}

]

}

}复制代码

每个 worker 都是一个单独的有 600ms 限制的 node.js 进程。同时跨进程的数据交换也会被限制。请在高开销的loader中使用，否则效果不佳

⑥压缩加速——开启多线程压缩

* 不推荐使用 webpack-paralle-uglify-plugin，项目基本处于没人维护的阶段，issue 没人处理，pr没人合并。  
  Webpack 4.0以前：uglifyjs-webpack-plugin，parallel参数

module.exports = {

optimization: {

minimizer: [

new UglifyJsPlugin({

parallel: true,

}),

],

},};复制代码

* 推荐使用 terser-webpack-plugin

module.exports = {

optimization: {

minimizer: [new TerserPlugin(

parallel: true // 多线程

)],

},

};复制代码

**2）优化 Webpack 的打包体积**

* 压缩代码
* 提取页面公共资源：
* Tree shaking
* Scope hoisting
* 图片压缩
* 动态Polyfill

**3）speed-measure-webpack-plugin**  
简称 SMP，分析出 Webpack 打包过程中 Loader 和 Plugin 的耗时，有助于找到构建过程中的性能瓶颈。 **开发阶段**

开启多核压缩 插件：\*\* terser-webpack-plugin \*\*

const TerserPlugin = require('terser-webpack-plugin')module.exports = {

optimization: {

minimizer: [

new TerserPlugin({

parallel: true,

terserOptions: {

ecma: 6,

},

}),

]

}

}复制代码

传送门 ☞[# 工程化专题](https://juejin.cn/post/6996188856164810789" \o "https://juejin.cn/post/6996188856164810789" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

## Babel

### 简单描述一下 Babel 的编译过程

Babel 是一个 JavaScript 编译器，是一个工具链，主要用于将采用 ECMAScript 2015+ 语法编写的代码转换为向后兼容的 JavaScript 语法，以便能够运行在当前和旧版本的浏览器或其他环境中。

Babel 本质上就是在操作 AST 来完成代码的转译。AST是抽象语法树（Abstract Syntax Tree, AST）

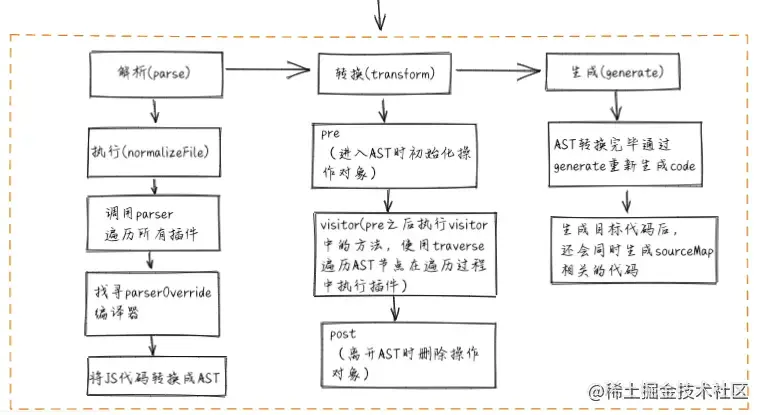
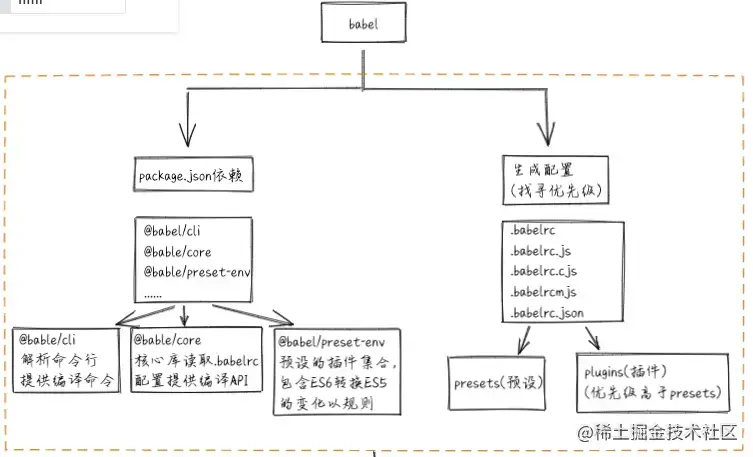
如果想要了解更多，可以阅读和尝试：

* 分析 AST：[ASTexplorer.net](https://link.juejin.cn?target=https://astexplorer.net/" \o "https://astexplorer.net/" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)
* AST 规范：[github.com/estree/estr…](https://link.juejin.cn?target=https://github.com/estree/estree" \o "https://github.com/estree/estree" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

Babel 的功能很纯粹，它只是一个编译器。大多数编译器的工作过程可以分为三部分：

1. **解析（Parse）** ：将源代码转换成更加抽象的表示方法（例如抽象语法树）。包括词法分析和语法分析。词法分析主要把字符流源代码（Char Stream）转换成令牌流（ Token Stream），语法分析主要是将令牌流转换成抽象语法树（Abstract Syntax Tree，AST）。
2. **转换（Transform）** ：通过 Babel 的插件能力，对（抽象语法树）做一些特殊处理，将高版本语法的 AST 转换成支持低版本语法的 AST。让它符合编译器的期望，当然在此过程中也可以对 AST 的 Node 节点进行优化操作，比如添加、更新以及移除节点等。
3. **生成（Generate）** ：将 AST 转换成字符串形式的低版本代码，同时也能创建 Source Map 映射。

经过这三个阶段，代码就被 Babel 转译成功了。

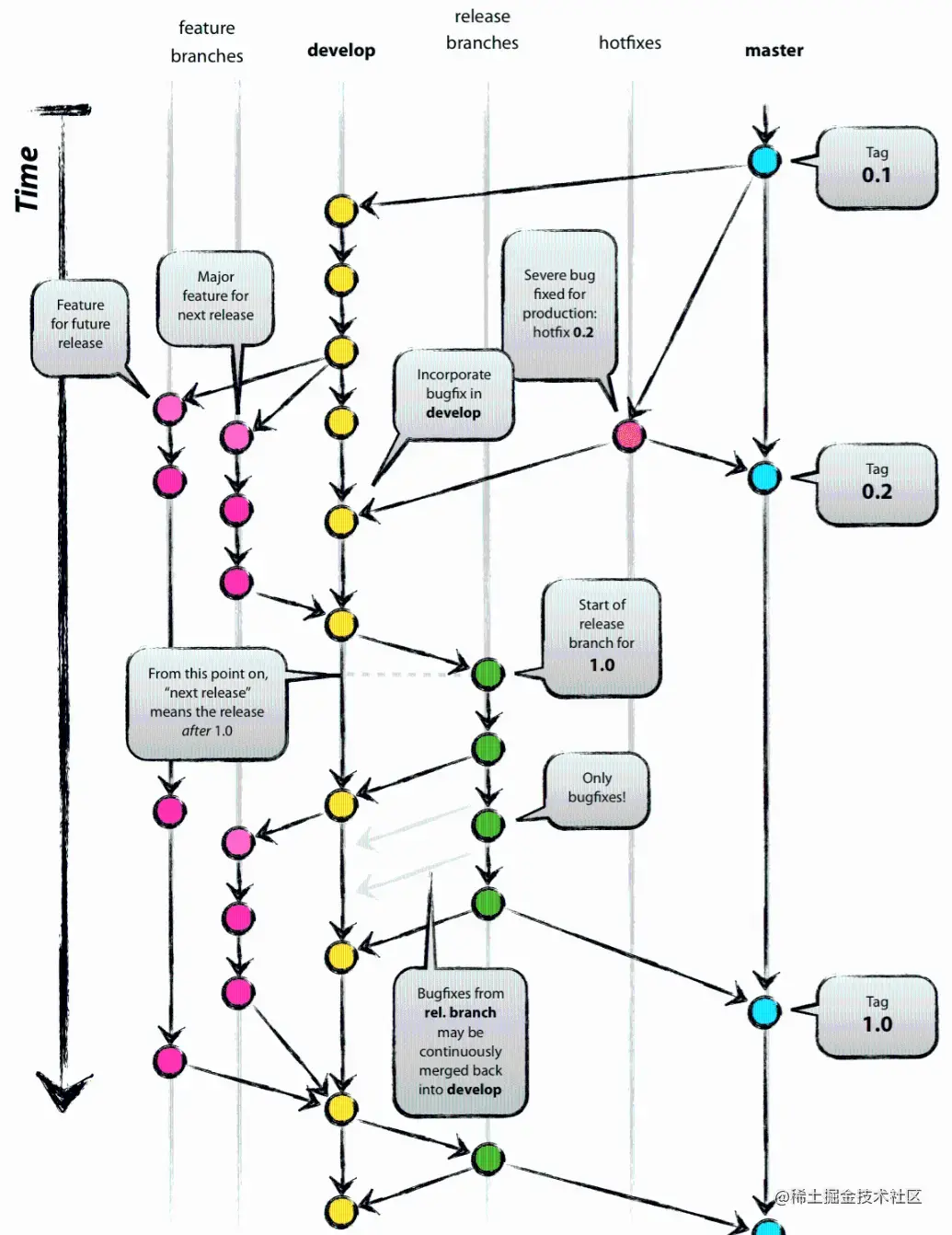


## Git

### Git 常用命令

查看分支：git branch  
创建分支：git branch  
切换分支：git checkout  
创建+切换分支：git checkout -b  
合并某分支到当前分支：git merge  
删除分支：git branch -d

### 如何使用Git管理项目



实际开发中，一个仓库（一般只放一个项目）主要存在两条主分支：master与develop分支。这个两个分支的生命周期是整个项目周期。

我们可能使用的不同类型的分支对项目进行管理是：

功能分支 功能分支（或有时称为主题分支）用于为即将发布或遥远的未来版本开发新功能。在开始开发某个功能时，将包含该功能的目标版本在那时很可能是未知的。功能分支的本质在于，只要该功能处于开发阶段，它就存在，但最终会被合并回develop（明确将新功能添加到即将发布的版本中）或丢弃。功能分支通常只存在于开发者仓库中，而不存在于origin。

发布分支 发布分支支持准备新的生产版本。它们允许在最后一刻打点 i 和交叉 t。此外，它们允许修复小错误并为发布准备元数据（版本号、构建日期等）。通过在发布分支上完成所有这些工作，该develop 分支被清除以接收下一个大版本的功能。

* + 从develop分支拉取，且必须合并回 develop 和 master
  + 分支命名约定：release-\*

修补程序分支

Hotfix 分支与发布分支非常相似，因为它们也旨在为新的生产版本做准备，尽管是计划外的。它们产生于需要立即对现场制作版本的不良状态采取行动。当必须立即解决生产版本中的关键错误时，可以从标记生产版本的主分支上的相应标记中分支出一个修补程序分支。

**master**：这个分支最为稳定，这个分支表明项目处于可发布的状态。  
**develop**：做为开发的分支，平行于master分支。  
**Feature branches**：这种分支和咱们程序员平常开发最为密切，称做功能分支。必须从develop分支建立，完成后合并回develop分支。  
**Release branches**：这个分支用来分布新版本。从develop分支建立，完成后合并回develop与master分支。这个分支上能够作一些很是小的bug修复，固然，你也能够禁止在这个分支作任何bug的修复工做，而只作版本发布的相关操做，例如设置版本号等操做，那样的话那些发现的小bug就必须放到下一个版本修复了。若是在这个分支上发现了大bug，那么也绝对不能在这个分支上改，须要Featrue分支上改，走正常的流程。  
**Hotfix branches**：这个分支主要为修复线上特别紧急的bug准备的。必须从master分支建立，完成后合并回develop与master分支。这个分支主要是解决线上版本的紧急bug修复的，例如忽然版本V0.1上有一个致命bug，必须修复。那么咱们就能够从master 分支上发布这个版本那个时间点 例如 tag v0.1（通常代码发布后会及时在master上打tag），来建立一个 hotfix-v0.1.1的分支，而后在这个分支上改bug，而后发布新的版本。最后将代码合并回develop与master分支。

[更多请参考](https://link.juejin.cn?target=https://nvie.com/posts/a-successful-git-branching-model/" \o "https://link.juejin.cn?target=https%3A%2F%2Fnvie.com%2Fposts%2Fa-successful-git-branching-model%2F" \t "C:\\Users\\86181\\AppData\\Local\\Temp\\_blank)

### 项目优化

移除生产环境的控制台打印。方案很多，esling+pre-commit、使用插件自动去除，插件包括babel-plugin-transform-remove-console、uglifyjs-webpack-plugin、terser-webpack-plugin。最后选择了terser-webpack-plugin，脚手架vue-cli用这个插件来开启缓存和多线程打包，无需安装额外的插件，仅需在configureWebpack中设置terser插件的drop\_console为true即可。最好还是养成良好的代码习惯，在开发基本完成后去掉无用的console，vscode中的turbo console就蛮好的。

第三方库的按需加载。echarts，官方文档里是使用配置文件指定使用的模块，另一种使用babel-plugin-equire实现按需加载。element-ui使用babel-plugin-component实现按需引入。

前后端数据交换方面，推动项目组使用蓝湖、接口文档，与后端同学协商，规范后台数据返回。

雅虎军规提到的，避免css表达式、滤镜，较少DOM操作，优化图片、精灵图，避免图片空链接等。

性能问题：页面加载性能、动画性能、操作性能。Performance API，记录性能数据。

winter重学前端 优化技术方案：

缓存：客户端控制的强缓存策略。

降低请求成本：DNS 由客户端控制，隔一段时间主动请求获取域名IP，不走系统DNS（完全看不懂）。TCP/TLS连接复用，服务器升级到HTTP2，尽量合并域名。

减少请求数：JS、CSS打包到HTML。JS控制图片异步加载、懒加载。小型图片使用data-uri。

较少传输体积：尽量使用SVG\gradient代替图片。根据机型和网络状况控制图片清晰度。对低清晰度图片使用锐化来提升体验。设计上避免大型背景图。

使用CDN加速，内容分发网络，是建立再承载网基础上的虚拟分布式网络，能够将源站内容缓存到全国或全球的节点服务器上。用户就近获取内容，提高了资源的访问速度，分担源站压力。

作者：Gaby  
链接：https://juejin.cn/post/7016593221815910408  
来源：稀土掘金  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。