作者：我很酷  
链接：https://zhuanlan.zhihu.com/p/158079643  
来源：知乎  
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。

**前言**

先赞再看，养成习惯~

网上参差不弃的面试题，本文由浅入深，让你在做面试官的时候，能够辨别出面试者是不是真的有点东西，也能让你去面试中级前端工程师更有底气。但是切记把背诵面试题当成了你的唯一求职方向

下篇将是非常硬核的源码，原理，自己编写框架和库等，如果感觉写得不错，可以关注给个star～

越是开放性的题目，更能体现回答者的水平，一场好的面试，不仅能发现面试者的不足，也能找到他的闪光点，还能提升面试官自身的技术

**1.Css和Html合并在第一个题目，请简述你让一个元素在窗口中消失以及垂直水平居中的方法，还有Flex布局的理解**

标准答案：百度上当然很多，这里不做阐述，好的回答思路是：

* 元素消失的方案先列出来， display:none和visibility: hidden;的区别，拓展到vue框架的v-if和v-show的区别，可以搭配回流和重绘来讲解

[我的官方群](https://link.zhihu.com/?target=https%3A//jq.qq.com/%3F_wv%3D1027%26k%3DkYVOL6ES)

**回流必将引起重绘，重绘不一定会引起回流**

**回流(Reflow)：**

当Render Tree中部分或全部元素的尺寸、结构、或某些属性发生改变时，浏览器重新渲染部分或全部文档的过程称为回流

* 下面内容会导致回流:
  + 页面首次渲染
  + 浏览器窗口大小发生改变
  + 元素尺寸或位置发生改变
  + 元素内容变化（文字数量或图片大小等等）
  + 元素字体大小变化
  + 添加或者删除可见的DOM元素
  + 激活CSS伪类（例如：:hover）
  + 查询某些属性或调用某些方法
* 一些常用且会导致回流的属性和方法：
  + clientWidth、clientHeight、clientTop、clientLeft
  + offsetWidth、offsetHeight、offsetTop、offsetLeft
  + scrollWidth、scrollHeight、scrollTop、scrollLeft
  + scrollIntoView()、scrollIntoViewIfNeeded()
  + getComputedStyle()
  + getBoundingClientRect()
  + scrollTo()

**重绘**

当页面中元素样式的改变并不影响它在文档流中的位置时（例如：color、background-color、visibility等），浏览器会将新样式赋予给元素并重新绘制它，这个过程称为重绘。

**性能影响对比：**



* 列出元素垂直居中的方案，以及各种方案的缺陷
* 讲出flex常用的场景，以及flex 1做了什么

上面的问题如果答得非常好，在重绘和回流这块要下大功夫。这点是前端性能优化的基础，而性能优化是前端最重要的核心基础技能点，也是面试官最看中的基础之一

**对于想要学习前端，但是不知道从何入手的小伙伴们，我特意准备了个qun，整理了一份最全面的前端学习资料（最基础的【HTML+css+JavaScript】到移动端的HTML5的项目实战教程视频），想学习的都可以加入**

[正在跳转​jq.qq.com](https://link.zhihu.com/?target=https%3A//jq.qq.com/%3F_wv%3D1027%26k%3DkYVOL6ES" \t "_blank)

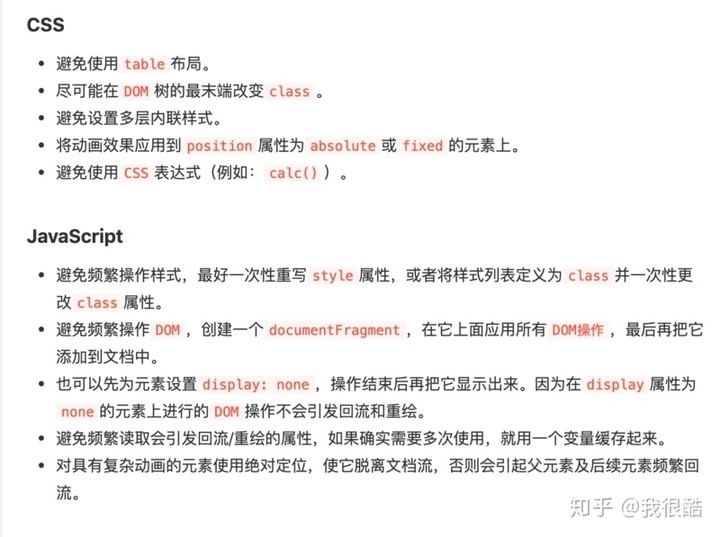
**喜欢的朋友还请点个star+关注~**

**2.你对This了解吗，有自己实现过call,apply,bind吗？**

href="[https://segmentfault.com/a/1190000020044435](https://link.zhihu.com/?target=https%3A//segmentfault.com/a/1190000020044435)">50行javaScript代码实现call,apply,bind

这是一个很基础的技能点，考察你对闭包，函数调用的理解程度，我感觉我写得比较简单容易懂

**3.如何减少重绘和回流的次数：**



**4.你对前端的异步编程有哪些了解呢**

这个题目如果回答非常完美，那么可以判断这个人已经脱离了初级前端工程师，前端的核心就是异步编程，这个题目也是体现前端工程师基础是否扎实的最重要依据。

还是老规矩，从易到难吧

**传统的定时器，异步编程：**

setTimeout(),setInterval()等。

缺点：当同步的代码比较多的时候，不确定异步定时器的任务时候能在指定的时间执行。

例如：

在第100行执行代码 setTimeout(()=>{console.log(1)},1000)//1s后执行里面函数

但是后面可能有10000行代码+很多计算的任务，例如循环遍历，那么1s后就无法输出console.log(1)

可能要到2s甚至更久

setInterval跟上面同理 当同步代码比较多时，不确保每次能在一样的间隔执行代码，

如果是动画，那么可能会掉帧

**ES6的异步编程：**

promise generator async

new promise((resolve,reject)=>{ resolve() }).then()....

缺点： 仍然没有摆脱回掉函数，虽然改善了回掉地狱

generator函数 调用next()执行到下一个yeild的代码内容，如果传入参数则作为上一个

`yield`的

返回值

缺点：不够自动化

async await

只有async函数内部可以用await,将异步代码变成同步书写，但是由于async函数本身返回一个

promise,也很容易产生async嵌套地狱

**requestAnimationFrame和requestIdleCallback：**

传统的javascript 动画是通过定时器 setTimeout 或者 setInterval 实现的。但是定时器动画一直存在两个问题

第一个就是动画的循时间环间隔不好确定，设置长了动画显得不够平滑流畅，设置短了浏览器的重绘频率会达到瓶颈，推荐的最佳循环间隔是17ms（大多数电脑的显示器刷新频率是60Hz，1000ms/60）；

第二个问题是定时器第二个时间参数只是指定了多久后将动画任务添加到浏览器的UI线程队列中，如果UI线程处于忙碌状态，那么动画不会立刻执行。为了解决这些问题，H5 中加入了 requestAnimationFrame以及requestIdleCallback

requestAnimationFrame 会把每一帧中的所有 DOM 操作集中起来，在一次重绘或回流中就完成，并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率

在隐藏或不可见的元素中，requestAnimationFrame 将不会进行重绘或回流，这当然就意味着更少的 CPU、GPU 和内存使用量

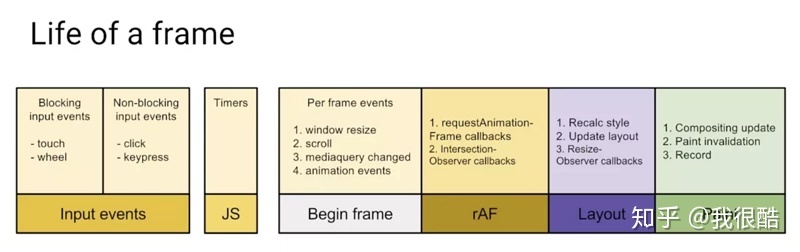
requestAnimationFrame 是由浏览器专门为动画提供的 API，在运行时浏览器会自动优化方法的调用，并且如果页面不是激活状态下的话，动画会自动暂停，有效节省了 CPU 开销

**性能对比：**



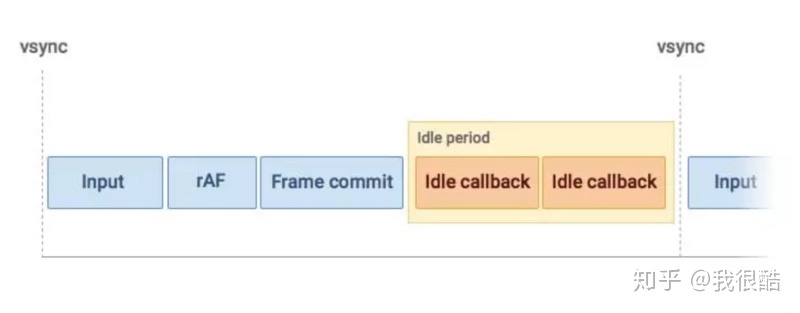
requestAnimationFrame的回调会在每一帧确定执行，属于高优先级任务，而requestIdleCallback的回调则不一定，属于低优先级任务。

我们所看到的网页，都是浏览器一帧一帧绘制出来的，通常认为FPS为60的时候是比较流畅的，而FPS为个位数的时候就属于用户可以感知到的卡顿了，那么在一帧里面浏览器都要做哪些事情呢，如下所示：



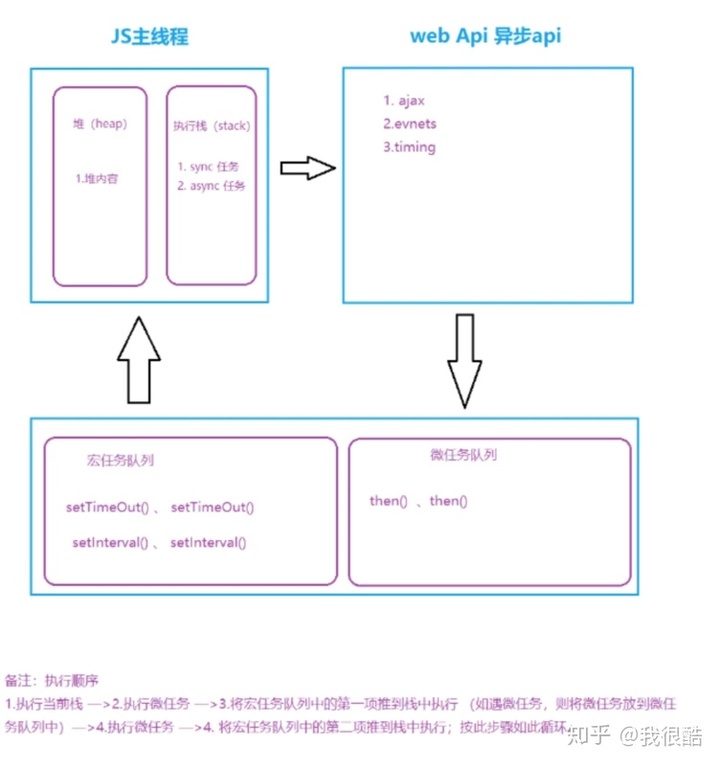
图中一帧包含了用户的交互、js的执行、以及requestAnimationFrame的调用，布局计算以及页面的重绘等工作。

假如某一帧里面要执行的任务不多，在不到16ms（1000/60)的时间内就完成了上述任务的话，那么这一帧就会有一定的空闲时间，这段时间就恰好可以用来执行requestIdleCallback的回调，如下图所示：



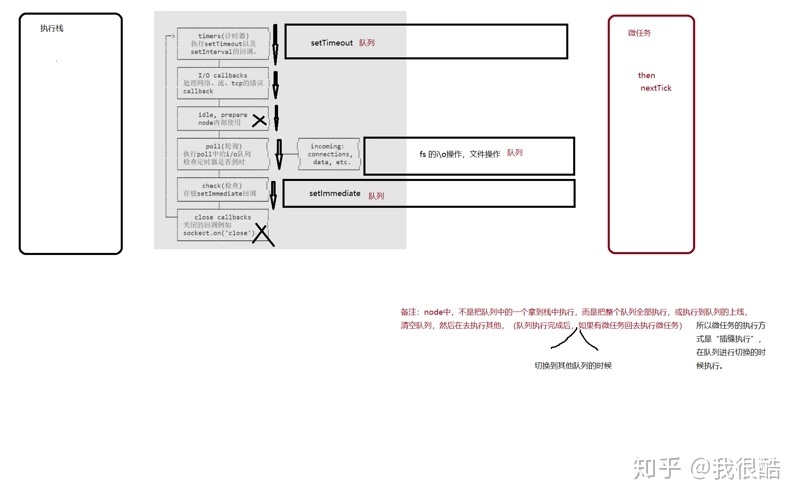
**5.简述浏览器的Eventloop和Node.js的Eventloop**

**浏览器的EventLoop**



不想解释太多，看图

**Node.js的EventLoop**



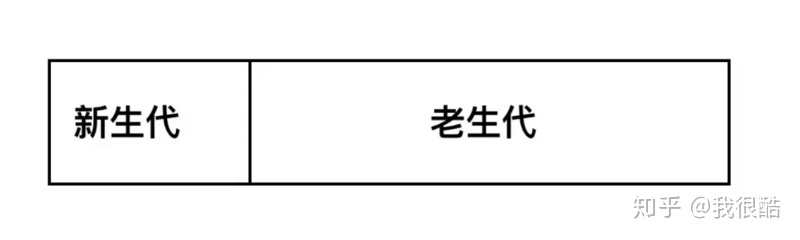
**特别提示：网上大部分Node.js的EventLoop的面试题，都会有BUG，代码量和计算量太少，很可能还没有执行到微任务的代码，定时器就到时间被执行了**

**6.闭包与V8垃圾回收机制：**

JS 的垃圾回收机制的基本原理是：

找出那些不再继续使用的变量，然后释放其占用的内存，垃圾收集器会按照固定的时间间隔周期性地执行这一操作。

V8 的垃圾回收策略主要基于分代式垃圾回收机制，在 V8 中，将内存分为新生代和老生代，新生代的对象为存活时间较短的对象，老生代的对象为存活事件较长或常驻内存的对象。



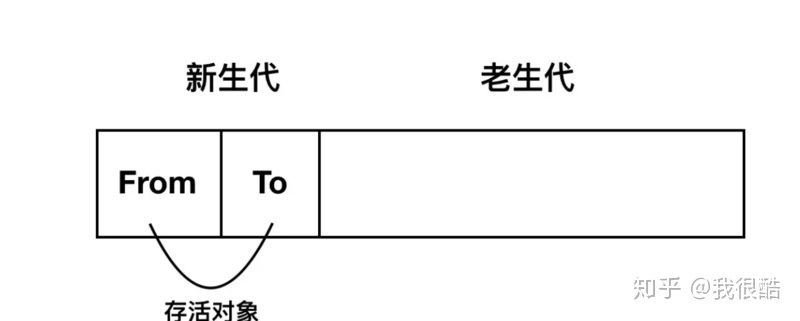
V8 堆的整体大小等于新生代所用内存空间加上老生代的内存空间，而只能在启动时指定，意味着运行时无法自动扩充，如果超过了极限值，就会引起进程出错。

**Scavenge 算法**

在分代的基础上，新生代的对象主要通过 Scavenge 算法进行垃圾回收，在 Scavenge 具体实现中，主要采用了一种复制的方式的方法—— Cheney 算法。

Cheney 算法将堆内存一分为二，一个处于使用状态的空间叫 From 空间，一个处于闲置状态的空间称为 To 空间。分配对象时，先是在 From 空间中进行分配。

当开始进行垃圾回收时，会检查 From 空间中的存活对象，将其复制到 To 空间中，而非存活对象占用的空间将会被释放。完成复制后，From 空间和 To 空间的角色发生对换。



当一个对象经过多次复制后依然存活，他将会被认为是生命周期较长的对象，随后会被移动到老生代中，采用新的算法进行管理。

还有一种情况是，如果复制一个对象到 To 空间时，To 空间占用超过了 25%，则这个对象会被直接晋升到老生代空间中。

**标记-清除和标记-整理算法**

对于老生代中的对象，主要采用标记-清除和标记-整理算法。标记-清除 和前文提到的标记一样，与 Scavenge 算法相比，标记清除不会将内存空间划为两半，标记清除在标记阶段会标记活着的对象，而在内存回收阶段，它会清除没有被标记的对象。

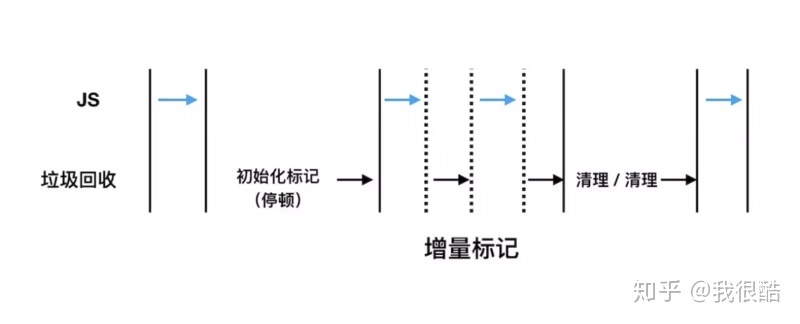
而标记整理是为了解决标记清除后留下的内存碎片问题。

**增量标记（Incremental Marking）算法**

前面的三种算法，都需要将正在执行的 JavaScript 应用逻辑暂停下来，待垃圾回收完毕后再恢复。这种行为叫作“全停顿”（stop-the-world）。

在 V8 新生代的分代回收中，只收集新生代，而新生代通常配置较小，且存活对象较少，所以全停顿的影响不大，而老生代就相反了。

为了降低全部老生代全堆垃圾回收带来的停顿时间，V8将标记过程分为一个个的子标记过程，同时让垃圾回收标记和JS应用逻辑交替进行，直到标记阶段完成。



经过增量标记改进后，垃圾回收的最大停顿时间可以减少到原来的 1/6 左右。

**内存泄漏**

内存泄漏（Memory Leak）是指程序中己动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

内存泄漏的常见场景：

* 缓存:存在内存中数据一只没有被清掉
* 作用域未释放（闭包）
* 无效的 DOM 引用
* 没必要的全局变量
* 定时器未清除(React中的合成事件，还有原生事件的绑定区别)
* 事件监听为清空
* 内存泄漏优化

**7.你熟悉哪些通信协议，它们的优缺点？**

[通信协议全解](https://link.zhihu.com/?target=https%3A//segmentfault.com/a/1190000019891825)

我的这篇文章非常详细介绍了 http1.0 http1.1 http2.0 https websocket等协议

**8.从输入url地址栏，发生了什么？由此来介绍如何性能优化：**

[性能优化不完全手册](https://link.zhihu.com/?target=https%3A//segmentfault.com/a/1190000018827395)

[如何优化你的超大型React应用](https://link.zhihu.com/?target=https%3A//juejin.im/post/5d35d3ecf265da1bc23fb654)

我的这两篇文章基本上涵盖了前端基础的性能优化，后期我会再出专栏。

**9.浏览器的缓存实现，请您介绍：**

1.preload,prefetch,dns-prefetch等

[什么是preload](https://link.zhihu.com/?target=https%3A//juejin.im/post/5b5984b851882561da216311)

使用 preload 指令的好处包括：

允许浏览器来设定资源加载的优先级因此可以允许前端开发者来优化指定资源的加载。

赋予浏览器决定资源类型的能力，因此它能分辨这个资源在以后是否可以重复利用。

浏览器可以通过指定 as 属性来决定这个请求是否符合 content security policy。

浏览器可以基于资源的类型（比如 image/webp）来发送适当的 accept 头。

Prefetch

Prefetch 是一个低优先级的资源提示，允许浏览器在后台（空闲时）获取将来可能用得到的资源，并且将他们存储在浏览器的缓存中。一旦一个页面加载完毕就会开始下载其他的资源，然后当用户点击了一个带有 prefetched 的连接，它将可以立刻从缓存中加载内容。

DNS Prefetching

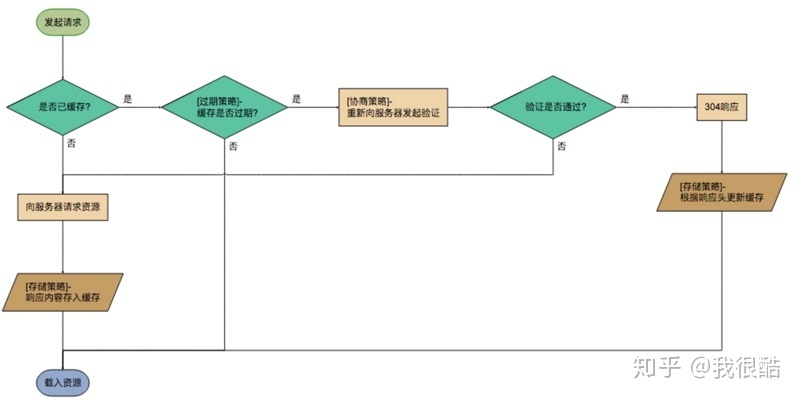
DNS prefetching 允许浏览器在用户浏览页面时在后台运行 DNS 的解析。如此一来，DNS 的解析在用户点击一个链接时已经完成，所以可以减少延迟。可以在一个 link 标签的属性中添加 rel="dns-prefetch' 来对指定的 URL 进行 DNS prefetching，我们建议对 Google fonts，Google Analytics 和 CDN 进行处理。

2.servece-worker,PWA渐进式web应用

[PWA文档](https://link.zhihu.com/?target=https%3A//lavas.baidu.com/pwa/README)



3.localstorage,sessionstorage,cookie,session等。  
[浏览器的会话存储和持久性存储](https://link.zhihu.com/?target=https%3A//juejin.im/post/5ad5b9116fb9a028e014fb19)  
4.浏览器缓存的实现机制的实现



**10.同源策略是什么，跨域解决办法，cookie可以跨域吗？**

[跨域解决的办法](https://link.zhihu.com/?target=https%3A//juejin.im/post/59c132415188256bb018e825)

Q：为什么会出现跨域问题？

A：出于浏览器的同源策略限制，浏览器会拒绝跨域请求。

* 注：严格的说，浏览器并不是拒绝所有的跨域请求，实际上拒绝的是跨域的读操作。浏览器的同源限制策略是这样执行的：

通常浏览器允许进行跨域写操作（Cross-origin writes），如链接，重定向；

通常浏览器允许跨域资源嵌入（Cross-origin embedding），如 img、script 标签；

通常浏览器不允许跨域读操作（Cross-origin reads）。

Q：什么情况才算作跨域？

A：非同源请求，均为跨域。名词解释：同源 —— 如果两个页面拥有相同的协议（protocol），端口（port）和主机（host），那么这两个页面就属于同一个源（origin）。

Q：为什么有跨域需求?

A：场景 —— 工程服务化后，不同职责的服务分散在不同的工程中，往往这些工程的域名是不同的，但一个需求可能需要对应到多个服务，这时便需要调用不同服务的接口，因此会出现跨域。

方法：JSONP,CORS,postmessage,webscoket，反向代理服务器等。