html

结构标记有哪些，他们与div有什么相同之处，又有什么不同之处

> header，nav，section，aside，footer，article
>
> 都是用来做布局的，但结构标记提高了语义性和可读性

说说HTML5，CSS3的新特性，平时使用过哪些

> H5新特性：
>
> 结构标记标签闭合、标签小写、不乱嵌套、提高搜索机器人搜索几率、使用外链 css 和 js 脚本、结构与行为表现的分离、文件下载与页面速度更快、内容能被更多的用户所访问、内容能被更广泛的设备所访问、更少的代码和组件，容易维护、改版方便，不需要变动页面内容、提供打印版本而不需要复制内容、提高网站易用性（header，footer，section，article，aside，nav），新表单元素（url，email，number，date，month，color等），canvas，audio，video，地理定位，拖放，w eb存储，Web Workers，WebSocket等。
>
> CSS3新特性：

复杂选择器器（属性选择器，伪类选择器，伪元素选择器，兄弟选择器等），边框倒角，边框阴影，渐变，转换，过渡，动画，弹性盒子，媒体查询技术等。

html5中新增特性

> 语义标签 、增强型表单 、视频和音频 、Canvas绘图 、SVG绘图 、地理定位 、拖放API
>
> WebWorker、WebStorage 、WebSocket

请简述代码<meta name=”viewport” content=” width=device-width,user-scalable=no,initial-scale=1.0,maximum-scale=1.0,minimum-scale=1.0” >中每个属性的含义

> **name=”viewport”** 设置视口
>
> 手机浏览器是把页面放在一个虚拟的“窗口”（viewport）中，通常这个虚拟的“窗口”（viewport）比屏幕宽，这样就不用把每个网页挤到很小的窗口中，也不会破坏没有针对手机浏览器优化的网页的布局，用户可以通过平移和缩放来看网页的不同部分。移动版的 Safari 浏览器最新引进了viewport 这个 meta tag，让网页开发者来控制 viewport 的大小和缩放，其他手机浏览器也基本支持。
>
> 通俗的讲，移动设备上的viewport就是设备的屏幕上能用来显示我们的网页的那一块区域，在具体一点，就是浏览器上(也可能是一个app中的webview)用来显示网页的那部分区域，但viewport又不局限于浏览器可视区域的大小，它可能比浏览器的可视区域要大，也可能比浏览器的可视区域要小。在默认情况

下，一般来讲，移动设备上的viewport都是要大于浏览器可视区域的，这是因为考虑到移动设备的分辨率相对于桌面电脑来说都比较小，所以为了能在移动设备上正常显示那些传统的为桌面浏览器设计的网站，移动设备上的浏览器都会把自己默认的viewport设为980px或1024px（也可能是其它值，这个是由设备自己决定的），但带来的后果就是浏览器会出现横向滚动条，因为浏览器可视区域的宽度是比这个默认的viewport的宽度要小的。

1、width ： 控制viewport的大小，可以指定一个值，如600， 或者特殊的值，如device-width为设备的宽度（单位是缩放为100%的CSS的像素）

2、height ： 和width相对应，指定高度

3、initial-scale ： 初始缩放比例，页面第一次加载时的缩放比例

4、maximum-scale ： 允许用户缩放到的最大比例，范围从0到10.0

5、minimum-scale ： 允许用户缩放到的最小比例，范围从0到10.0

6、user-scalable ： 用户是否可以手动缩放，值可以是：①yes、 true允许用户缩放；②no、false不允许用户缩放

iframe有那些缺点

> iframe会阻塞主页面的Onload事件；
>
> iframe和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。使用iframe之前需要考虑这两个缺点。如果需要使用iframe，最好是通过javascript动态给iframe添加src属性值，这

> 样可以可以绕开以上两个问题。

src和href的区别

> （1）href：href是Hypertext Reference的缩写，表示超文本引用。用来建立当前元素和文档之间的链接。并行下载该文档，不会停止对当前文档的处理
>
> （2）src： src是source的缩写，src指向的内容会嵌入到文档当前标签所在位置。如img、script、iframe当浏览器解析到该元素时，会暂停浏览器的渲染，直到该资源加载完毕。

CSS

为什么要初始化 CSS 样式

> 因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对 CSS 初始化往往会出现浏览器之间的页面显示差异。

描述css reset的作用和用途

> Reset重置浏览器的css默认属性 浏览器的品种不同，样式不同，然后重置，让他们统一

CSS选择器优先级

> !important(>1000)，内联样式(>1000)，id选择器(100)，(伪)类选择器(10)，标签选择器(1)，*选择器(0)。如果优先级相同，则选择最后出现的样式。

px 和 em 的区别

> px 和 em 都是长度单位，区别是，px 的值是固定的，指定是多少就是多少，计算比较容易。
>
> em 得值不是固定的，并且 em 会继承父级元素的字体大小。
>
> 浏览器的默认字体高都是 16px。所以未经调整的浏览器都符合: 1em=16px。那么 12px=0.75em, 10px=0.625em

说说em和rem的区别

> rem是基于html元素的字体大小来决定，
>
> 而em则根据使用它的元素的字体大小决定（很多人错误以为是根据父类元素，实际上是元素继承了父类的属性才会产生的错觉）

rbga和opcity的区别

>

> opacity属性的值，可以被其子元素继承，给父级div设置opacity属性，那么所有子元素都会继承这个属性，并且，该元素及其继承该属性的所有子元素的所有内容透明度都会改变。而RGBA设置的元素，只对该元素的背景色有改变，并且，该元素的后代不会继承该属性

display有哪些值，说明他们的作用

> display的取值可以是none/inline/block/inline-block/table
>
> none:元素隐藏不可见，并且元素也不占页面空间。
>
> Inline:元素将呈现行内元素的特点，多个元素在一行中显示，不允许修改尺寸，也不能设置垂直外边距。
>
> block:元素将呈现块级元素的特点，每个元素独占一行，允许修改尺寸。
>
> inline-block:元素将呈现行内块元素的特点，多个元素在一行中显示，但是可以修改尺寸。
>
> table:元素将呈现表格的特点，由内容决定表格的显示方式。元素独占一行，允许修改尺寸。

隐藏网页中的元素有几种方式这些方法有什么区别

> 三种方法
>
> display:none；脱离文档流，不占页面空间,会改变页面布局。
>
> visivility:hidden ；不会脱离文档流，不会改变页面布局，仍占页面空间

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

盒模型默认的计算方式是什么要改变默认的计算方式用哪个属性及值

> 实际占地宽度=左右外边距+左右边框+左右内边距+width
>
> 实际占地高度=上下外边距+上下边框+上下内边距+height
>
> 改变计算方式　box-sizing:border-box

如何解决子元素的上外边距溢出

> 在父元素的第一个子元素位置处添加一个空的table标记

浮动会对父元素的高度带来什么影响如何解决这个问题

> 高度坍塌　，　万能清除

清除浮动带来影响的几种方式，各自的优缺点

> 1、直接设置父元素高度，但不是每次都知道父元素的高度。
>
> 2、设置父元素也浮动，但不是任何时候父元素都需要浮动，而且浮动会影响后续元素。

3、为父元素设置 overflow属性，但如果有内容需要溢出显示的话，也会一同被隐藏。

4、在父元素中，追加空子元素，并设置其clear:both,但页面中会多出一个空元素。

5、用内容生成的方式:after{content:"";display:block;clear:both;}

css有哪些定位

> 固定定位、相对定位、绝对定位、粘性定位

改变页面元素位置的定位方式及他们的区别

> relative 相对定位 相对于自身原来位置进行偏移，仍处于标准文档流中；
>
> absolute 绝对定位 相对于最近的已定位的祖先元素，有已定位(指position不是static的元素)祖先元素，以最近的祖先元素为参考标准。如果无已定位祖先元素，以body元素为偏移参照， 完全脱离文档流；
>
> fixed 固定定位的元素会相对于视窗来定位,这意味着即便页面滚动，它还是会停留在固定的位置。固定定位也会脱离文档流。

说出link和import的区别

> （1）使用方法不同：
>
> link一般在html头部定义，作为html标签，链接资源，主要用于链接外部的样式表

> import一般定义在css内部，并且一定要在其他所有规则之前，也就是一般写在文件头部，并且专门拿来链css的。
>
> （2）加载顺序差别
>
> import比link延迟一些，它会等到页面下载完后才加载，因而可能会产生闪烁

CSS3的flexbox：

> “
> 该布局模型的目的是提供一种更加高效的方式来对容器中的条目进行布局、对齐和分配空间。在传统的布局方式中，block 布局是把块在垂直方向从上到下依次排列的；而 inline 布局则是在水平方向来排列。弹性盒布局并没有这样内在的方向限制，可以由开发人员自由操作。
>
> 试用场景：弹性布局适合于移动前端开发，在Android和ios上也完美支持。

垂直居中的方法

> “
> 1、定位+外边距 盒子宽高已知，
>
> position: absolute; left: 50%; top: 50%; margin-left:-自身一半宽度; margin-top: -自身一半高度;
>
> 2、定位+transform 盒子高宽已知，
>
> #container{ position:relative; border:1px solid red; width:800px; height:600px; }
>
> #center{ width:100px; height:100px; background:blue; position:absolute; top:50%; left:50%; transform: translate(-50%,-50%); }

3、flex 布局

父级：

display: flex;

align-items: center;

justify-content: center;

css sprites，如何使用。

> Css 精灵 把一堆小的图片整合到一张大的图片上，减轻服务器对图片的请求数量

PC端页面怎么适应不同分辨率

> 媒体查询

简单解释下bfc

> BFC 即 Block Formatting Contexts (块级格式化上下文)
>
> BFC 是Block Formatting Context 的简写，翻译为块级格式上下文。它会创建一个特殊的区域，在这个区域中，只有block box 参与布局。而BFC的一套特点和规则就规定了在这个特殊的区域中如何尽心布局，如何进行定位，区域内元素的相互关系和相互作用。这个特殊的区域不受外界影响。
>
> 上面提到了block box 的概念，block box 是指display属性为 block、list-item、table 的元素。

相应地，我们有inline block ，它是指 display 属性为inline,inline,inline-table 的元素。css3 规范中又加入了 run in box。

如何形成BFC

根元素或其他包含它的元素

浮动元素（元素的float不是none）

绝对定位元素（元素具有 position为absolute或fixed）

内联块（元素具有 display ： inline-block）

表格单元格（元素具有 display: table-cell,html表格单元格默认属性）

表格标题（元素具有 display:table-caption，html表格标题默认属性）

具有overflow且值不是 visible 的块元素

display:flow-root 的元素，这是唯一没有副作用的一种

column-span: all 的元素

BFC决定了什么

内部的box将会独占宽度，且在垂直方向，一个接一个排列

box垂直方向的间距由 margin 属性决定，但是同一个 BFC 的两个相邻box的margin 会出现边距折叠现象

每个box水平方向上左边缘，与BFC左边缘相对齐，即使存在浮动也是如此

BFC 区域不会与浮动元素重叠，而是会依次排列

> BFC区域内是一个独立的渲染容器，容器内元素和BFC区域外元素不会形成任何干扰
>
> 浮动元素的高度也参与到BFC高度的计算当中
>
> **BFC 解决了什么问题**
>
> 父元素高度塌陷
>
> 外边距折叠
>
> 自适应多栏布局

移动端怎么适配

> viewport 适配
>
> vw 适配（部分等比缩放）
>
> rem 适配
>
> 弹性盒适配（合理布局）

浏览器渲染页面的过程

> 1.根据html文件构建DOM树和CSSOM树。构建DOM树期间，如果遇到JS，阻塞DOM树及CSSOM树的构建，优先加载JS文件，加载完毕，再继续构建DOM树及CSSOM树。
>
> 2.构建渲染树（Render Tree）。
>
> 3.页面的重绘（repaint）与重排（reflow，也有称回流）。页面渲染完成后，若JS操作了DOM节点，根据JS对DOM操作动作的大小，浏览器对页面进行重绘或是重排。

重排重绘的理解

> **重绘（repaint）**：屏幕的一部分要重绘。渲染树节点发生改变，但不影响该节点在页面当中的空间位置及大小。譬如某个div标签节点的背景颜色、字体颜色等等发生改变，但是该div标签节点的宽、高、内外边距并不发生变化，此时触发浏览器重绘（repaint）。
>
> **重排（reflow）**：也有称回流，当渲染树节点发生改变，影响了节点的几何属性（如宽、高、内边距、外边距、或是float、position、display：none;等等），导致节点位置发生变化，此时触发浏览器重排（reflow），需要重新生成渲染树。譬如JS为某个p标签节点添加新的样式："display:none;"。导致该p标签被隐藏起来，该p标签之后的所有节点位置都会发生改变。此时浏览器需要重新生成渲染树，重新布局，即重排（reflow）。
>
> 注意：**重排必将引起重绘，而重绘不一定会引起重排。**

何时回引起重排

> 当页面布局和几何属性改变时就需要重排。下述情况会发生浏览器重排：
>
> 1、添加或者删除可见的DOM元素；
>
> 2、元素位置改变——display、float、position、overflow等等；
>
> 3、元素尺寸改变——边距、填充、边框、宽度和高度

4、内容改变——比如文本改变或者图片大小改变而引起的计算值宽度和高度改变；

5、页面渲染初始化；

6、浏览器窗口尺寸改变——resize事件发生时；

Canvas和SVG的区别是什么

> canvas绘制2d位图svg绘制2d 矢量图
>
> canvas通过js程序指令创建图形，svg通过标签创建图形
>
> canvas可以只能将事件绑定在画布上，svg可以将事件绑定在任意元素上
>
> canvas应用方向，网页特效与小游戏，svg创建统计图与地图应用

animation 和 transition 的区别

> 1.transition 是过渡，是样式值的变化的过程，只有开始和结束；animation 其实也叫关键帧，通过和 keyframe 结合可以设置中间帧的一个状态；
>
> 2.animation 配合 @keyframe 可以不触发时间就触发这个过程，而 transition 需要通过 hover 或者 js 事件来配合触发；
>
> 3.animation 可以设置很多的属性，比如循环次数，动画结束的状态等等，transition 只能触发一次；
>
> 4.animation 可以结合 keyframe 设置每一帧，但是 transition 只有两帧；

5.在性能方面：浏览器有一个主线程和排版线程；主线程一般是对 js 运行的、页面布局、生成位图等等，然后把生成好的位图传递给排版线程，而排版线程会通过 GPU 将位图绘制到页面上，也会向主线程请求位图等等；我们在用使用 aniamtion 的时候这样就可以改变很多属性，像我们改变了 width、height、postion 等等这些改变文档流的属性的时候就会引起，页面的回流和重绘，对性能影响就比较大，但是我们用 transition 的时候一般会结合 tansfrom 来进行旋转和缩放等不会生成新的位图，当然也就不会引起页面的重排了

在scss中如何定义混合器以及引用

> @mixin 混合器名称{ 样式声明 }
>
> 通过@include来使用混合器。

浏览器兼容性常用方式

> css兼容：cssrset、加浏览器前缀、css hack

http

简述浏览器打开 www.baidu.com 显示页面，中间都经过哪些过程

> 向DNS服务器获取域名对应的IP地址并返回浏览器，通过IP地址向web服务器发请求，web服务器从文件服务器获取网页中所需要的文件，从数据库服务器获取网页中所需要的数据，最后讲完整的网页相应给浏览器

什么是异步数据加载

> 浏览器在向服务器发送请求的时候，不耽误用户在网页上做其它操作，可以同时开启多个任务，并且可以以无刷新的效果来更改页面中的局部内容

异步请求数据的步骤分为哪几步

> 创建异步对象，绑定监听事件，创建异步请求，发送异步请求

异步请求中满足什么条件时才能取出响应的结果数据

> 请求状态码为4，响应状态码为200时

http状态码

> 200 - 请求成功
>
> 301 - 资源（网页等）被永久转移到其它URL
>
> 302 - 资源（网页等）临时转移到其它URL

304 - 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源

400 (错误请求) 服务器不理解请求的语法。

403 (禁止) 服务器拒绝请求。

404 - 请求的资源（网页等）不存在

500 - 内部服务器错误

什么是ajax有哪些优点

> AJAX是“Asynchronous JavaScript and XML”的缩写。他是指一种创建交互式网页应用的网页开发技术。
>
> 优点：
>
> 1.局部刷新页面,减少用户心理和实际的等待时间,带来更好的用户体验。
>
> 2.减轻服务器的压力,按需取数据,最大程度的减少冗余数据请求。
>
> 3.基于xml标准化,并被广泛支持,不需安装插件。
>
> 4.促进页面和数据的分离。

同源策略是什么

> 同源是指"协议+域名+端口"三者完全相同

JSONP 的原理是什么

> 尽管浏览器有同源策略，但是 script 标签的 src 属性不会被同源策略所约束，可以 获取任意服务器上的脚本并执行。jsonp 通过插入 script 标签的方式来实现跨域，参数只能通过 url 传入，仅能支持 get 请求

你如何对网站的文件和资源进行优化

> 文件合并
>
> 文件最小化/文件压缩
>
> 使用 CDN 托管
>
> 缓存的使用

常用http的http方法有哪些

> GET:用于请求访问已经被uri识别的资源,可以通过url传参给服务器.
>
> POST:用于传输信息给服务器,主要功能GET类似,但一般推荐使用POST方式.
>
> PUT:传输文件,报文主体中包含文件内容,保存到对应的uri位置.
>
> DELETE:删除文件,与PUT方法相反,删除对应URI位置的文件.
>
> OPTIONS:查询对应uri支持的http方法
>
> HEAD:获得报文首部,与GET方法类似,只是不返回报文主体,一般用于验证uri是否有效.

GET和POST的区别

> 区别一:
>
> GET重点从服务器上获取资源,POST重点向服务器发送数据.
>
> 区别二:
>
> get传输数据是通过URL请求，置于URL后，多个请求数据间用”&”连接.post传输数据通过Http的post机制，将字段与对应值封存在请求实体中发送给服务器，这个过程对用户是不可见的;
>
> 区别三:
>
> Get传输的数据量小，因为受URL长度限制，但效率较高;
>
> Post可以传输大量数据，所以上传文件时只能用Post方式;
>
> 区别四:
>
> get是不安全的，因为URL是可见的，可能会泄露私密信息，如密码等;
>
> post较get安全性较高;
>
> 区别五:
>
> get方式只能支持ASCII字符，向服务器传的中文字符可能会乱码。
>
> post支持标准字符集，可以正确传递中文字符。

HTTP请求报文与响应报文格式

> 请求报文包含三部分：
>
> a、请求行：包含请求方法、URI、HTTP版本信息
>
> b、请求首部字段
>
> c、请求内容实体
>
> 响应报文包含三部分：
>
> a、状态行：包含HTTP版本、状态码、状态码的原因短语
>
> b、响应首部字段
>
> c、响应内容实体

http和https区别和优化

> HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。简单来说，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。
>
> HTTPS和HTTP的区别主要如下：
>
> 1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。
>
> 2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。
>
> 3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

网络请求性能优化

“

出发点：1，减少请求次数。2，简化请求过程。3，压缩请求数据

方法：

1，减少DNS查找：每次主机名的解析都需要一次网络往返，从而增加了请求的延迟时间，同时还会阻塞后续的请求。

2，减少HTTP重定向。HTTP冲定向需要额外的DNS查询、TCP握手等非常耗时，最佳的重定向次数为0.

3，使用CDN（内容分发网络）：把数据放在离用户地理位置更近的地方，可以明显减少每次TCP连接的网络延迟，增大吞吐量。

4，删除没有必要请求的资源。

5，在客户端缓存资源：缓存必要的应用资源，避免每次都重复请求相同的内容，例如多图片下载可以考虑使用缓存。

6，内容在传输前先压缩：传输数据之前应该先压缩应用资源，把要传输的字节减少到最小，在压缩的时候确保对每种不同的资源采用最好的压缩手段。

7，消除不必要的请求开销：减少请求的HTTP首部数据（比如HTTP cookie）.

JS

continue和 break有什么区别

> break和continue都是用来控制循环结构的：
>
> break终止循环，跳出循环体执行循环后面的语句。
>
> continue跳过本次循环，执行下一次循环。

i++和++i的区别

> i++ ： 先用i值后加1，
>
> ++i ： 先加1后用i值，

JavaScript都有哪些数据类型

> 7种原始类型：undefined、Boolean、Numbe、String、、BigInt、Symbol、null
>
> 1种引用类型：Object

取 1~11之间的随机数（即包括1不包括11）

> Math.floor(Math.random()*10+1)
>
> parseInt（Math.random()*10+1)

什么是变量声明提前

> 使用var关键字声明的变量，会将声明提升到所在作用域的最前边

作用域链的理解

> 由多级作用域串联形成的链式结构

null和undefined的区别

> null表示"没有对象"，即该处不应该有值
>
> undefined表示"缺少值"，就是此处应该有一个值，但是还没有定义
>
> null和undefined转换成number数据类型：null 默认转成 0、undefined 默认转成 NaN

==和===有什么不同

> == 抽象相等，比较时，会先进行类型转换，然后再比较值
>
> ===严格相等，判断两个值相等同时数据类型也得相同

setTimeout和setInterval的区别是什么

> 二者都是用来设置定时操作的

setTimeout: 设置一个定时器，在定时器到期后执行一次函数或代码段

setInterval: 设置一个定时器，以固定的时间间隔重复调用一个函数或者代码段

var，let，const有什么区别

> const必须初始化，而且不能更改
>
> var仅在function是局部变量，其余都为全局变量；
>
> let一直充当局部变量,即便是在if中，外界也不能访问
>
> 函数变量提升，提升也就是把定义的放在一段代码的最开头
>
> var 的声明，存在变量提升。let 的声明，不存在变量提升。const 的声明，不存在变量提升。
>
> for循环中，var定义相当于在for这一片区是全局的，所以值是最后一个；let是每个独立分开的，所以值不同
>
> var可以重复声明，let不可以

如何判断 js 中的数据类型

> typeof、instanceof
>
> 使用 typeof 操作符
>
> typeof 不适合用于判断是否为数组当使用 typeof 判断数组和对象的时候，都会返回 object。 可以使用 isArray()来判断是否为数组
>
> Instanceof

> instanceof 只能用来判断对象和函数，不能用来判断字符串和数字等。判断它是否为字符串和 数字时，只会返回 false

为什么NaN不等于NaN

> NaN不是一个具体的值，而是表示某个值不是数字

闭包：

> 一个函数和对其周围状态（lexical environment，词法环境）的引用捆绑在一起（或者说函数被引用包围），这样的组合就是闭包（closure）。也就是说，闭包让你可以在一个内层函数中访问到其外层函数的作用域。

使用箭头函数应注意什么

> 用了箭头函数，this 就不是指向 window，而是父级（指向是可变的）
>
> 不能够使用 arguments 对象
>
> 不能用作构造函数，这就是说不能够使用 new 命令，否则会抛出一个错误

如何理解JSON

> JSON是JS对象的一种表现方式，即以js对象的数据格式表现出来的字符串，JSON中的两个api如下：

将JSON字符串转换成JSON对象 JSON.parse()

将JSON对象转换成JSON字符串 JSON.stringify()

forEach和map的区别:

> map方法返回一个新的数组，数组中的元素为原始数组调用函数处理后的值。map方法不会对空数组进行检测，map方法不会改变原始数组。
>
> forEach方法用来调用数组的每个元素，将元素传给回调函数。forEach对于空数组是不会调用回调函数的。 无论arr是不是空数组，forEach返回的都是undefined。这个方法只是将数组中的每一项作为callback的参数执行一次。

apply,call,bind 的区别

> apply: 函数中的 this 替换成参数 1，其余参数放数组中. 直接触发函数
>
> call: 函数中的 this 替换成参数 1，其余参数依次摆放. 直接触发函数
>
> bind: 替换函数中的 this 指向 并 传入其他参数，返回新的函数. 不会直接触发函数
>
> 改变 this 指向的方法 apply,call,bind

同步和异步有何区别

> 同步：指发送一个请求,需要等待返回,然后才能够发送下一个请求，有等待过程（在一个任务进行中时不能开启其他的任务）。

异步：指发送一个请求,不需要等待返回,随时可以再发送下一个请求，即不需要等待（在一个任务进行中时可以开启其他的任务）。

介绍下 promise 的特性、优缺点

> Promise 基本特性
>
> 1、Promise 有三种状态：pending(进行中)、fulfilled(已成功)、rejected(已失败)
>
> 2、Promise 对象接受一个回调函数作为参数，该回调函数接受两个参数，分别是成 功时的回调 resolve 和失败时的回调 reject；另外resolve 的参数除了正常值以外，还可能是一个 Promise 对象的实例；reject 的参数通常是一个 Error 对象的实例。
>
> 3、then 方法返回一个新的 Promise 实例，并接收两个参数 onResolved(fulfilled 状 态的回调); onRejected(rejected 状态的回调，该参数可选)4、catch 方法返回一个新的 Promise 实例
>
> 5、finally 方法不管 Promise 状态如何都会执行，该方法的回调函数不接受任何参数
>
> 6、Promise.all()方法将多个多个 Promise 实例，包装成一个新的 Promise 实例，该 方法接受一个由 Promise 对象组成的数组作为参数(Promise.all()方法的参数可以不 是数组，但必须具有 Iterator 接口，且返回的每个成员都是 Promise 实例)，注意参 数中只要有一个实例触发 catch 方法，都会触发 Promise.all()方法返回的新的实例的 catch 方法，如果参数中的某个实例本身调用了 catch 方法，将不会触发 Promise.all() 方法返回的新实例的 catch 方法

7、Promise.race()方法的参数与 Promise.all 方法一样，参数中的实例只要有一个率 先改变状态就会将该实例的状态传给 Promise.race()方法，并将返回值作为 Promise.race()方法产生的 Promise 实例的返回值

8、Promise.resolve()将现有对象转为 Promise 对象，如果该方法的参数为一个 Promise 对象，Promise.resolve()将不做任何处理；如果参数 thenable 对象(即具有 then 方法)，Promise.resolve()将该对象转为 Promise 对象并立即执行 then 方法；

如果参数是一个原始值，或者是一个不具有 then 方法的对象，则 Promise.resolve 方法返回一个新的 Promise 对象，状态为 fulfilled，其参数将会作为 then 方法中 onResolved 回调函数的参数，如果 Promise.resolve 方法不带参数，会直接返回一 个 fulfilled 状态的 Promise 对象。需要注意的是，立即 resolve()的 Promise 对象， 是在本轮“事件循环”（event loop）的结束时执行，而不是在下一轮“事件循环”的开 始时。

9、Promise.reject()同样返回一个新的 Promise 对象，状态为 rejected，无论传入任 何参数都将作为 reject()的参数

Promise 缺点

“

1、无法取消 Promise，一旦新建它就会立即执行，无法中途取消。

2、如果不设置回调函数，Promise 内部抛出的错误，不会反应到外部

3、当处于 Pending 状态时，无法得知目前进展到哪一个阶段

4、Promise 真正执行回调的时候，定义 Promise 那部分实际上已经走完了，所以

5、Promise 的报错堆栈上下文不太友好

promise与async 有什么区别async的返回值是什么

> 1 promise是ES6，async/await是ES7
>
> 2 async/await相对于promise来讲，写法更加优雅
>
> 3 reject状态
>
> 1）promise错误可以通过catch来捕捉，建议尾部捕获错误
>
> 2）async/await既可以用.then又可以用try-catch捕捉
>
> async函数返回一个Promise对象

Object 结构 和 Map 解构的区别

> Object 结构提供了“字符串—值”的对应
>
> Map 结构提供了“值—值”的对应，是一种更完善的 Hash 结构实现
>
> 如果你需要“键值对”的数据结构，Map 比 Object 更合适

Set 和 Map 的使用

> 1.Set 用法：ES6 提供了新的数据结构 Set，它类似于数组，但是成员的值都是唯一的，没有重复的值
>
> 2.Map 用法：ES6 提供了 Map 数据结构。它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。

介绍 JavaScript 的原型，原型链有什么特点

> 原型：
>
> - JavaScript 的所有对象中都包含了一个 [proto] 内部属性，这个属性所对应的就是该对象的原型
>
> - JavaScript 的函数对象，除了原型 [proto] 之外，还预置了 prototype 属性
>
> - 当函数对象作为构造函数创建实例时，该 prototype 属性值将被作为实例对象的原型 [proto]。
>
> 原型链：
>
> - 当一个对象调用的属性/方法自身不存在时，就会去自己 [proto] 关联的前辈 prototype 对象上去找
>
> - 如果没找到，就会去该 prototype 原型 [proto] 关联的前辈 prototype 去找。依次类推，直到找到属性/方法或 undefined 为止。从而形成了所谓的“原型链”
>
> 原型特点：
>
> - JavaScript 对象是通过引用来传递的，当修改原型时，与之相关的对象也会继承这一改变

说说严格模式的限制

> 严格模式主要有以下限制：
>
> - 变量必须声明后再使用
> - 函数的参数不能有同名属性，否则报错
> - 不能使用 with 语句
> - 不能对只读属性赋值，否则报错
> - 不能使用前缀 0 表示八进制数，否则报错
> - 不能删除不可删除的属性，否则报错
> - 不能删除变量 delete prop，会报错，只能删除属性 delete global[prop]
> - eval 不会在它的外层作用域引入变量
> - eval 和 arguments 不能被重新赋值
> - arguments 不会自动反映函数参数的变化
> - 不能使用 arguments.callee
> - 不能使用 arguments.caller
> - 禁止 this 指向全局对象
> - 不能使用 fn.caller 和 fn.arguments 获取函数调用的堆栈
> - 增加了保留字（比如 protected、static 和 interface）

ES6的语法

> 箭头函数、解构赋值、函数参数默认值、模板字符串、let 、const、promise

那些操作会造成内存泄漏

> 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在
>
> setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏
>
> 闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

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
>
> 非立即执行版的意思是触发事件后函数不会立即执行，而是在 n 秒后执行，如果在 n 秒内又
>
> 触发了事件，则会重新计算函数执行时间。
>
> 立即执行版：
>
> 立即执行版的意思是触发事件后函数会立即执行，然后 n 秒内不触发事件才能继续执行函数的
>
> 效果。
>
> 在开发过程中，我们需要根据不同的场景来决定我们需要使用哪一个版本的防抖函数，一般来
>
> 讲上述的防抖函数都能满足大部分的场景需求。但我们也可以将非立即执行版和立即执行版的

> 防抖函数结合起来，实现最终的双剑合璧版的防抖函数

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。
>
> 时间戳版：在持续触发事件的过程中，函数会立即执行，并且每 1s 执行一次。
>
> 定时器版:在持续触发事件的过程中，函数不会立即执行，并且每 1s 执行一次，在停止触发事
>
> 件后，函数还会再执行一次。
>
> 时间戳版和定时器版的节流函数的区别就是，时间戳版的函数触发是在时间段内开始的时候，
>
> 而定时器版的函数触发是在时间段内结束的时候

谈谈Javascript 垃圾回收方法

> “
> 标记清除（mark and sweep）
>
> - 这是 JavaScript 最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”

- 垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

引用计数(reference counting)

- 在低版本 IE 中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个 变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加 1，如果该变量的值变成了另外一个，则这个值得引用次数减 1，当这个值的引用次数变为 0 的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为 0 的值占用的空间

深拷贝浅拷贝是什么意思

> 浅拷贝是会将对象的每个属性进行依次复制，但是当对象的属性值是引用类型时，实质复制的是其引用，当引用指向的值改变时也会跟着变化
>
> 深拷贝复制变量值，对于引用数据，则递归至基本类型后，再复制
>
> 深拷贝后的对象与原来的对象是完全隔离的，互不影响，对一个对象的修改并不会影响另一个对象

dom

按HTML查找和按选择器查找的差别

> 1.返回值不同：按HTML查找返回动态集合，按选择器查找返回非动态集合
>
> 2.效率不同：按HTML查找效率高，按选择器查找效率低
>
> 3.易用性不同：当条件复杂时，按HTML查找繁琐，而按选择器查找简单

事件传播的三个阶段是什么

> 捕获 > 目标 > 冒泡；在捕获阶段，事件通过父元素向下传递到目标元素。 然后
>
> 它到达目标元素，冒泡开始。

利用冒泡和不利用冒泡的差别

> 1.绑定位置不同：不利用冒泡绑定在目标元素上，利用冒泡绑定在父元素上
>
> 2.监听对象的个数不同：不利用冒泡会反复创建多个监听，利用冒泡始终只有一个监听
>
> 3.动态生成的元素：不利用冒泡无法自动获得事件处理函数，必须反复绑定
>
> 利用冒泡可让动态添加的子元素自动获得父元素的处理函数，无需反复绑定

谈谈事件委托/代理

>

> 事件委托是指将事件绑定目标元素的到父元素上，利用冒泡机制触发该事件
>
> 优点：
>
> - 可以减少事件注册，节省大量内存占用
> - 可以将事件应用于动态添加的子元素上
>
> 缺点： 使用不当会造成事件在不应该触发时触发

IE 的事件处理和 W3C 的事件处理有哪些区别

> 绑定事件
>
> - W3C: targetEl.addEventListener('click', handler, false);
> - IE: targetEl.attachEvent('onclick', handler);
>
> 删除事件
>
> - W3C: targetEl.removeEventListener('click', handler, false);
> - IE: targetEl.detachEvent(event, handler);
>
> 事件对象
>
> - W3C: var e = arguments.callee.caller.arguments[0]
> - IE: window.event
>
> 事件目标
>
> - W3C: e.target
> - IE: window.event.srcElement

阻止事件默认行为

- W3C: e.preventDefault()
- IE: window.event.returnValue = false'

阻止事件传播

- W3C: e.stopPropagation()
- IE: window.event.cancelBubble = true

IE和DOM事件流的区别

> 1.执行顺序不一样、
>
> 2.参数不一样
>
> 3.事件加不加on
>
> 4.this指向问题

事件是什么IE与火狐的事件机制有什么区别 如何阻止冒泡

> （1） 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。
>
> （2） 事件处理机制：IE是事件冒泡、火狐是 事件捕获；
>
> （3） ev.stopPropagation();

.IE和标准下有哪些兼容性的写法

> Var ev = ev || window.event
> document.documentElement.clientWidth ||
> document.body.clientWidth
>
> Var target = ev.srcElement||ev.target

列举DOM中常用优化

> 1.查找时，如果之用一个条件就可查询出结果时，优先选择按HTML查找。如果查找条件复杂，则优先选择易用的按选择器查找
>
> 2.添加时，尽量减少操作DOM树的次数，减少重排重绘。如果同时添加父元素和子元素，应先将子元素添加到到父元素，最后再将父元素添加到DOM树。如果添加多个平级子元素，则应先将子元素添加到文档片段，最后，再将文档片段添加到DOM树
>
> 3.修改时，尽量减少重排重绘。如果同时修改多个元素的内容或样式，应尽量少使用innerHTML和cssText方式修改元素的内容和样式。应使用class批量修改样式
>
> 4.事件绑定时，应尽量利用冒泡减少事件监听的个数。

为按钮绑定事件，实现事件节流

> var canClick=true;
>
> btn.onclick=function(){
>
> if(canClick){
>
> canClick=false;

```
        console.log("发送ajax请求，加载更多...");
        setTimeout(function(){
          console.log("请求完成!");
          canClick=true;
        },3000);
}
}
```

为页面绑定滚动事件，实现事件防抖

```
var  timer1;
window.onscroll=function(){
if(timer1!==undefined){
        clearTimeout(timer1);
}
timer1=setTimeout(function(){
        console.log("发送ajax请求，加载更多");
        timer1=undefined;
},200)
}
```

如何最小化重绘(repaint)和回流(reflow)

> - 需要要对元素进行复杂的操作时，可以先隐藏(display:"none")，操作完成后再显示
>
> - 需要创建多个 DOM 节点时，使用 DocumentFragment 创建完后一次性的加入 document
>
> - 缓存 Layout 属性值，如：var left = elem.offsetLeft; 这样，多次使用 left 只产生一次回流
>
> - 尽量避免用 table 布局（table 元素一旦触发回流就会导致 table 里所有的其它元素回流）
>
> - 避免使用 css 表达式(expression)，因为每次调用都会重新计算值（包括加载页面）
>
> - 尽量使用 css 属性简写，如：用 border 代替 border-width，border-style，border-color
>
> 批量修改元素样式：elem.className 和 elem.style.cssText 代替 elem.style.xxx

前端优化的方法

> 减少dom操作
>
> 2.部署前，图片压缩，代码压缩
>
> 3.优化js代码结构，减少冗余代码
>
> 4.减少http请求，合理设置 HTTP缓存
>
> 5.使用内容分发cdn加速
>
> 6.静态资源缓存
>
> 7.图片延迟加载

解释jsonp的原理，以及为什么不是真正的ajax

> 利用元素的这个开放策略，网页可以得到从其他来源动态产生的JSON资料，而这种使用模式就是所谓的JSONP。用JSONP抓到的资料并不是JSON，而是任意的javaScript，用javaScript直译器执行而不是用JSON解析器解析。
>
> ajax的核心是通过XmlHttpRequest获取非本页内容，而jsonp的核心则是动态添加标签来调用服务器提供的js脚本。

请解释一下 JavaScript 的同源策略，为什么要有同源限制

> 概念:同源策略是客户端脚本（尤其是 Javascript）的重要的安全度量标准。它最早出自 Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议
>
> - 指一段脚本只能读取来自同一来源的窗口和文档的属性
>
> 我们举例说明：比如一个黑客程序，他利用 Iframe 把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过 Javascript 读取到你的表单中 input 中的内容，这样用户名，密码就轻松到手了。]
>
> 缺点: 现在网站的 JS 都会进行压缩，一些文件用了严格模式，而另一些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节

实现跨域访问有几种方式

> 主要有两种
>
> JSONP:
>
> 客户端: 客户端动态添加script元素，用script发送请求，代替ajax请求，并携带客户端一个函数名到服务端
>
> 服务端: 接收客户端发来的函数名，将函数名和要返回的数据拼接为一条可执行的js语句，返回
>
> CORS: Cross-Origin Resources Sharing
>
> 客户端正常发送ajax请求，服务端定义响应头，允许指定来源的请求跨域访问:
>
> res.writeHead(200,{
>
> …,
>
> “Access-Control-Allow-Origin”:”允许的请求来源域名”
>
> })

jquery

$的原理

> $(“选择器”）是先查找DOM元素，再将DOM元素放入jQuery对象中
>
> 其中自带优化:
>
> 如果选择器是#id，则自动调用getElementById
>
> 如果选择器是.class，则自动调用getElementsByClassName
>
> 如果选择器是标签名，则自动调用getElementsByTagName
>
> 否则，其它选择器，都自动调用querySelectorAll()
>
> $(DOM元素）是直接将DOM元素放入jQuery对象中
>
> $(“HTML片段”）是创建一个新元素
>
> $(function(){}）是绑定事件，在DOM内容加载后就提前触发。

实现动画有几种方式，哪种好

> CSS: transition，animateion
>
> 优点: 由专门的排版引擎解析，效率高
>
> 缺点: 无法随意控制交互行为
>
> JS: 定时器， $().animate()
>
> 优点: 可随意控制交互行为
>
> 缺点: 效率不如css动画
>
> requestAnimationFrame()
>
> 优点: 可根据浏览器的刷新频率自动优化动画效果

缺点: 新API，有兼容性问题

$(document).ready() 是个什么函数为什么要用它

> ready() 函数用于在文档进入ready状态时执行代码。当DOM 完全加载（例如HTML被完全解析DOM树构建完成时），jQuery允许你执行代码。使用$(document).ready()的最大好处在于它适用于所有浏览器，jQuery帮你解决了跨浏览器的难题。

JavaScript window.onload 事件和 jQuery ready 函数有何不同

> JavaScript window.onload 事件和 jQuery ready 函数之间的主要区别是，前者除了要等待 DOM 被创建还要等到包括大型图片、音频、视频在内的所有外部资源都完全加载。如果加载图片和媒体内容花费了大量时间，用户就会感受到定义在 window.onload 事件上的代码在执行时有明显的延迟。
>
> 另一方面，jQuery ready() 函数只需对 DOM 树的等待，而无需对图像或外部资源加载的等待，从而执行起来更快。使用 jQuery $(document).ready() 的另一个优势是你可以在网页里多次使用它，浏览器会按它们在 HTML 页面里出现的顺序执行它们，相反对于 onload 技术而言，只能在单一函数里使用。鉴于这个好处，用 jQuery ready() 函数比用 JavaScript window.onload 事件要更好些。

jQuery 的优点

> 核心理念：写的更少，做的更多。（write less,do more）。
>
> 轻量级，标准版和 mini 版，mini 版简化了函数的名称，减小了源码的大小。
>
> 语法便捷，可提高开发效率。
>
> 不用在 html 中插入一堆 js 来调用命令了，只需要定义 id 即可。

jQuery 对象

> jQuery 对象就是通过 jQuery 包装 DOM 对象后产生的对象。
>
> 但是需要注意：jQuery 对象虽然是包装 DOM 对象产生的，但是 jQuery 无法使用
>
> DOM 对象的任何方法，同理 DOM 对象也不调用 jQuery 的方法。

选择器

> jQuery 提供的强大选择器，分为：
>
> 1、基本选择器
>
> 2、层次选择器
>
> 3、基本过滤选择器
>
> 4、内容过滤选择器
>
> 5、可见度过滤选择器

6、属性选择器

7、子元素选择器

8、表单选择器

9、表单对象属性选择器

nodejs

什么是NodeJS

> Nodejs是一个JavaScript的运行环境，是一个服务器端的“JavaScript解释器”，用于方便高效地搭建一些响应速度快、易于扩展的网络应用。它采用事件驱动、异步编程，为网络服务而设计

NodeJS中有哪些类型模块

> 模块类型： 核心模块、自定义模块、第三方模块。

fs是什么模块

> fs模块是Node.js的一个核心模块,专门用来操作系统中的文件,常用的操作方式是对文件的读取和写入

使用require('fs')载入fs模块，模块中所有方法都有同步和异步两种形式。

异步是通过回调函数获取结果，同步是通过方法的返回值获取结果。

NodeJS中导入模块和导入js文件写法上有什么区别

> 引入模块，直接使用名字导入即可
>
> 导入js文件，需要使用文件的路径

请指出JavaScript宿主对象和原生对象的区别

> 宿主对象：指JavaScript解释器提供的对象，由解释器厂家自定义并提供实现，不同的解释器提供的扩展对象存在较大的差异（DOM和BOM对象）。
>
> 原生对象：JavaScript语言本身预定义的对象，在ECMAScript标准中定义，由所有的解释器厂家来提供具体实现
> （Array,Date,Math,Number,String,Boolean等）。

npm常用命令

> npm init
>
> npm install 模块名
>
> npm uninstall 模块名
>
> npm install -g 模块名
>
> npm install 模块名 -D

```
npm  install
```

简述常见的package.json配置项

> （1）name：这个很好理解，就是package的名称。不过需要注意的是，name有长度限制(虽然一般都不会超)，而且name不能以 【点】 或者 【下划线】 开头，name中不能有大写字母。这个是每一个package必须的。在业务代码中，通过require(${name})就可以引入对应的程序包了
>
> （2）version：package的版本。对于业务项目来说，这个往往不太重要，但是如果你要发布自己的项目，这个就显得十分重要了。name和version共同决定了唯一一份代码。npm是用[npm-semver来解析版本号的。我们一般见到的都是大版本.次要版本.小版本这种版本号，比如16.1.0
>
> （3）desription：包的描述。开发组件库时必需，简明的向库的使用者介绍这个库是干嘛的。对于公司的业务项目，这个配置项一般无所谓
>
> （4）keywords：关键词。一个字符串数组，对这个npm包的介绍。组件库必需，便于使用者在npm中搜索。对于公司业务项目，这个配置一般无所谓
>
> （5）homepage： 项目主页。对于开发组件库来说挺有用的
>
> （6）bugs：开发者的联系方式，代码库的issues地址等。如果代码使用者发现了bug，可以通过这个配置项找到提bug的地方
>
> （7）icense：开源协议。
>
> （8）author：项目的作者。可以为字符串，对象
>
> （9）contributors：项目的贡献者。author的数组

（10）main：代码入口。这个十分重要，特别是对于组件库。当你想在node_modules中修改你使用的某个组件库的代码时，首先在node_modules中找到这个组件库，第一眼就是要看这个main，找到组件库的入口文件。在这个入口文件中再去修改代码吧

（11）scripts：指定了运行脚本命令的npm命令行缩写。十分重要

（12）directories：对整个代码结构的描述。告诉代码包使用者可以在哪里找到对应的文件

（13）files：数组。表示代码包下载安装完成时包括的所有文件

（14）repository：对于组件库很有用。让组件库使用者找到你的代码库地址。这个配置项会直接在组件库的npm首页生效

（15）config：用于添加命令行的环境变量

（16）dependencies：项目的依赖。通过npm install --save安装的包会出现在这里。注意，不要把测（17）devDependencies：项目的依赖。通过npm run install --save-dev安装的包会出现在这里。主要是在开发过程中依赖的一些工具。用法与dependencies相似

（18）bundledDependencies：数组，打包时的依赖。如果配置了bundledDependencies，在项目中执行 npm pack将项目打包时，最后生成的.tgz包中，会包含bundledDependencies中配置的依赖。bundledDependencies中的依赖必须在devDependencies或者dependencies中声明过

（19）peerDependencies: 指定当前组件的依赖以其版本。如果组件使用者在项目中安装了其他版本的同一依赖，会提示报错

（20）engines：指定项目所依赖的node环境、npm版本等

（21）private：如果设为true，无法通过npm publish发布代码

（22）bin：用来指定各个内部命令对应的可执行文件的路径

对NodeJS的优点和缺点提出自己的看法

> CPU密集型任务的特点是进行大量的计算，消耗CPU资源，比如计算圆周率(上千位)、对视频进行编码等，全靠CPU的运算能力（一般用C语言，java）
>
> IO（Input / Output）密集型任务，这类任务的特点是CPU消耗很少，大部分时间都在等待IO操作。常见的大部分任务都是IO密集型任务，比如Web应用（一般用脚本语言：python/Nodejs）。
>
> Nodejs设计思想中以事件驱动、异步、非堵塞I/O密集型为核心，他提供的大多数api都是基于事件的、异步的风格。所以非常适合处理高并发请求。此外，与Node服务器交互的客户端代码是由js语言编写的，因此客户端和服务器端都用同一种语言编写，减少了成本。

vue

请说出vue.cli项目中src目录每个文件夹和文件的用法

> 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。

assets和static的区别

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

v-if 和 v-show 区别

> (1).手段：v-if是动态的向DOM树内添加或者删除DOM元素；v-show是通过设置DOM元素的display样式属性控制显隐；
>
> (2).编译过程：v-if切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件；v-show只是简单的基于css切换；
>
> (3).编译条件：v-if是惰性的，如果初始条件为假，则什么也不做；只有在条件第一次变为真时才开始局部编译（编译被缓存编译被缓存后，然后再切换的时候进行局部卸载)；v-show是在任何条件下（首次条件是否为真）都被编译，然后被缓存，而且DOM元素保留；
>
> (4).性能消耗：v-if有更高的切换消耗；v-show有更高的初始渲染消耗；
>
> (5).使用场景：v-if适合运营条件不大可能改变；v-show适合频繁切换。

如何让CSS只在当前组件中起作用

> 在组件中的 style 前面加上 scoped

的作用是什么

> 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。

说说你对Vue中keep-alive的理解

> keep-alive是一个抽象组件：它自身不会渲染一个DOM元素，也不会出现在父组件链中；使用keep-alive 包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。
>
> 用户在某个列表页面选择筛选条件过滤出一份数据列表，由列表页面进入数据详情页面，再返回该列表页面，我 们希望：列表页面可以保留用户的筛选（或选中）状态。 keep-alive就是用来解决这种场景。当然keep-alive不仅仅是能够保存页面/组件的状态这么简单，它还可以 避免组件反复创建和渲染，有效提升系统性能。总的来说，keep-alive用于保存组件的渲染状态

keep-alive的两种函数的作用

> .vue的一个内置组件，缓存组件内部状态，避免重新渲染
>
> 1.include - 字符串或正则表达式。只有匹配的组件会被缓存。
> 2.exclude - 字符串或正则表达式。任何匹配的组件都不会被缓存。

vue中的 ref 是什么

> ref 被用来给元素或子组件注册引用信息。引用信息将会注册在父组件的 $refs 对象上。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。

如何获取dom

> ref="domName" 用法：this.$refs.domName

说出几种vue当中的指令和它的用法

> 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。

v-modal的使用

> 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。

vue中 key 值的作用

> 当 Vue.js 用 v-for 正在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0p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。key的作用主要是为了高效的更新虚拟DOM。

methods、computed、watch三者的区别

> methods是个方法，执行的时候需要事件进行触发
>
> computed是一个计算属性，是实时响应的，只要data中的属性发生了变化那么就会触发computed，计算属 性是基于属性的依赖进行缓存的，methods调用的时候需要加()，而computed调用的时候是不需要加()
>
> watch属性监听，watch用来监听属性的变化，当值发生变化的时候来执行特定的函数，watch监听属性的时候 会有2个参数newVal和oldVal一个新值一个旧值

分别简述computed和watch的使用场景

> · 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

谈谈vue计算属性的好处

>

v-on可以监听多个方法吗

> 可以，栗子：。

$nextTick的使用

> 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。

路由跳转回退，滚动到浏览器上次访问位址

> 使用vue2 中的keep-alive缓存组件

vue组件中data为什么必须是一个函数

> 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。

Vue中双向数据绑定是如何实现的

> 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。

v-if和v-for的优先级

>

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。

什么是js的冒泡Vue中如何阻止冒泡事件

“
js冒泡概念：当父元素内多级子元素绑定了同一个事件，js会依次从内往外或者从外往内（）执行每个元素的该事件，从而引发冒泡

js解决冒泡：event.stopPropagation()

vue解决冒泡： 事件.stop,例如：@click.stop="" ,@mouseover.stop=""

vue常用的修饰符

“
· .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。

引进组件的步骤

> 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。

Vue里面router-link在电脑上有用，在安卓上没反应怎么解决

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

Vue2中注册在router-link上事件无效解决方法

> 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。

params和query的区别

> 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。

怎么定义 vue-router 的动态路由 怎么获取传过来的值

> 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。

vue-router 有哪几种导航钩子

> 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

vue-router 有哪几种导航钩子

> 全局导航钩子
>
> router.beforeEach(to，from，next),
>
> router.beforeResolve(to，from，next),
>
> router.afterEach(to，from ,next)
>
> 组件内钩子
>
> beforeRouteEnter,
>
> beforeRouteUpdate,
>
> beforeRouteLeave
>
> 单独路由独享组件
>
> beforeEnter

`$route` 和 `$router` 的区别

> $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。

vue-router的两种模式

> 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）。

vue-router实现路由懒加载 （ 动态加载路由 ）

> 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。

RouterLink在IE和Firefox中不起作用（路由不跳转）的问题

> 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

Vue 如何去除url中的#

> 使用 history 模式
>
> 需要注意的是，当我们启用 history 模式的时候，由于我们的项目是一个单页面应用，所以在路由跳转的时候，就会出现访问不到静态资源而出现 404 的情况，这时候就需要服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面

非父子组件间的数据传递，兄弟组件传值

> eventBus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。项目比较小时，用这个比较合适。（虽然也有不少人推荐直接用VUEX，具体来说看需求咯。技术只是手段，目的达到才是王道。）

vue初始化页面闪动问题

> 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 }"

delete和Vue.delete删除数组的区别

> 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。

vue相关的组件之间的通信方式

> 1.父组件传给子组件
>
> 2.子组件传给父组件
>
> 3.建立新的VUE对象
>
> 4.使用vuex
>
> 5.使用全局函数

什么是 vue 生命周期有什么作用

> 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。
>
> 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。

Vue生命周期的各个阶段作用

> **beforeCreate**（创建前） 在数据观测和初始化事件还未开始
> **created**（创建后） 完成数据观测，属性和方法的运算，初始化事件，$el属性还没有显示出来
> **beforeMount**（载入前） 在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，把data里面的数据和模板生成html。注意此时还没有挂载html到页面上。
> **mounted**（载入后） 在el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译好的html内容替换el属性指向的DOM对象。完成模板中的html渲染到html页面中。此过程中进行ajax交互。
> **beforeUpdate**（更新前） 在数据更新之前调用，发生在虚拟DOM重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。
> **updated**（更新后） 在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。调用时，组件DOM已经更新，所以可以执行依赖于DOM的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。
> **beforeDestroy**（销毁前） 在实例销毁之前调用。实例仍然完全可用。
> **destroyed**（销毁后） 在实例销毁之后调用。调用

后，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

第一次页面加载会触发哪几个钩子

> beforeCreate，created，beforeMount，mounted

created和mounted的区别

> created:在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图。mounted:在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作。

vue获取数据在哪个周期函数

> 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.

简述每个周期具体适合哪些场景

> 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。

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

在什么阶段才能访问操作DOM

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

axios及安装

> 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。

Vue.js中ajax请求代码应该写在组件的methods中还是vuex的actions中

> 如果请求来的数据是不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入vuex 的state里。
>
> 如果被其他地方复用，这个很大几率上是需要的，如果需要，请将请求放入action里，方便复用。

axios的特点有哪些

> 1、axios是一个基于promise的HTTP库，支持promise的所有API；
>
> 2、它可以拦截请求和响应；
>
> 3、它可以转换请求数据和响应数据，并对响应回来的内容自动转换为json类型的数据；
>
> 4、它安全性更高，客户端支持防御XSRF；

什么是跨域

> 跨域是由浏览器的同源策略造成的 同源策略：指域名、协议、端口相同；如果违反了同源策略就会造成跨域
>
> 跨域的解决 •
>
> cors 一般都是后端搞定，我们只需要顾虑前端的东西
>
> • jsonp 了解过，使用DOM操作动态的添加一个元素，而script标签中的src属性是 没有跨域限制的也就是逃避了同源策略，而且只能用get请求
>
> • 接口代理 通过代理服务器配置来实现，需要在vue.config.js 中配置poxy文件

请说下封装 vue 组件的过程

> 1.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。)

2.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。

3.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。

4.封装完毕了，直接调用即可

vuex有哪几种属性

> 有五种，分别是 State、 Getter、Mutation 、Action、 Module
>
> state为单一状态树，在state中需要定义我们所需要管理的数组、对象、字符串等等，只有在这里定义了，在vue.js的组件中才能获取你定义的这个对象的状态。
>
> getter有点类似vue.js的计算属性，当我们需要从store的state中派生出一些状态，那么我们就需要使用getter，getter会接收state作为第一个参数，而且getter的返回值会根据它的依赖被缓存起来，只有getter中的依赖值（state中的某个需要派生状态的值）发生改变的时候才会被重新计算。
>
> 更改store中state状态的唯一方法就是提交mutation，就很类似事件。每个mutation都有一个字符串类型的事件类型和一个回调函数，我们需要改变state的值就要在回调函数中改变。我们要执行这个回调函数，那么我们需要执行一个相应的调用方法：store.commit。
>
> action可以提交mutation，在action中可以执行store.commit，而且action中可以有任何的异步操作。在页面中如果我们要嗲用这个action，则需要执行store.dispatch

module其实只是解决了当state中很复杂臃肿的时候，module可以将store分割成模块，每个模块中拥有自己的state、mutation、action和getter。

不用Vuex会带来什么问题

> 一、可维护性会下降，你要想修改数据，你得维护三个地方
>
> 二、可读性会下降，因为一个组件里的数据，你根本就看不出来是从哪来的
>
> 三、增加耦合，大量的上传派发，会让耦合性大大的增加，本来Vue用Component就是为了减少耦合，现在这么用，和组件化的初衷相背。
>
> 但兄弟组件有大量通信的，建议一定要用，不管大项目和小项目，因为这样会省很多事

解释一下vuex

> vuex是用来做状态管理的，具有五个常用属性state，getter，actions，mutations，modules。state是数据源，类似vue中的data，我们可以通过两种方式来获取它，mapStates，mapGerters。并且获取state必须放到computed中这样能保证state发生改变的时候该组件中用到state的地方都发生变化。

页面刷新vuex被清空解决办法

> 1.localStorage 存储到本地再回去
>
> 2.重新获取接口获取数据

简述vuex的数据传递流程

> 当组件进行数据修改的时候我们需要调用dispatch来触发actions里面的方法。actions里面的每个方法中都会 有一个commit方法，当方法执行的时候会通过commit来触发mutations里面的方法进行数据的修改。 mutations里面的每个函数都会有一个state参数，这样就可以在mutations里面进行state的数据修改 ，当数据修改完毕后，会传导给页面。页面的数据也会发生改变

VueX原理

> vuex实现了一个单项数据流，在全局又有一个state存放数据，
>
> 当组建要更改state中的数据时，必须通过Mutation进行，mutation同时提供了订阅者模式供外部插件调用获 取state数据的更新。
>
> 而当所有异步操作（常见于调用后台接口异步获取更新数据）或批量的同步操作需要走Action，
>
> 但Action也是无法直接修改state的，还是需要通过mutation来修改state的数据。
>
> 后根据state的变化，渲染到视图上。

vue的双向绑定原理

> 1.vue数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的，数据和视图同步更新
>
> 2.核心：Object.defineProperty()

Object.defineProperty（）来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调。

对项目进行估哪些优化

> 1.懒加载
>
> 2.v-if v-show区分使用场景
>
> 3.v-for加key和不加key的区别

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
>
> （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

mvc与mvvm

> MVC模型 - 视图 - 控制器（Model-View-Controller）
>
> Model和View永远不能相互通信，只能通过Controller传递。
>
> Controller可以直接与Model对话（读写调用Model），Model通过Notification和KVO机制与Controller间接通信。
>
> Controller可以直接与View对话，通过outlet,直接操作View,outlet直接对应到View中的控件,View通过action向Controller报告事件的发生(如用户Touch我了)。Controller是View的直接数据源（数据很可能是Controller从Model中取得并经过加工了）。

Controller是View的代理（delegate),以同步View与Controller。

MVVM Model - ViewModel - View

什么是 MVVM：一个 MVC 的增强版，我们正式连接了视图和控制器，并将表示逻辑从 Controller 移出放到一个新的对象里，即 View Model。MVVM 听起来很复杂，但它本质上就是一个精心优化的 MVC 架构

Model层是少不了的了，我们得有东西充当DTO(数据传输对象)，当然，用字典也是可以的，编程么，要灵活一些。Model层是比较薄的一层，如果学过Java的小伙伴的话，对JavaBean应该不陌生吧。

ViewModel层，就是View和Model层的粘合剂，他是一个放置用户输入验证逻辑，视图显示逻辑，发起网络请求和其他各种各样的代码的极好的地方。说白了，就是把原来ViewController层的业务逻辑和页面逻辑等剥离出来放到ViewModel层。

View层，就是ViewController层，他的任务就是从ViewModel层获取数据，然后显示。

Vue的双向数据绑定原理是什么

> 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，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。
>
> 具体步骤：
>
> 第一步：需要observe的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter和getter

这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

第二步：compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

第三步：Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是:

1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个update()方法

3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

vue优点

“

· 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。

什么是单页应用

> 单页应用的全称是 Single Page Application Single Page Application，简称 SPA SPA
>
> 通过路由的变更，局部切换网页内容 取代 整个页面的刷新操作.
>
> 三大框架 React React Vue Vue Angular Angular 均采用单页应用模式.
>
> 优点: 1. 用户操作体验好，用户不用刷新页面。 2. 局部更新，对服务器压力小. 3. 良好的前后端分离. 后端不再负责页面渲染和输出工作. 缺点: 1. 首次加载耗时长，速度慢. 2. SEO SEO不友好，需要采用 prerender prerender 服务进行完善

单页面应用和多页面应用区别及优缺点

> 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。

vue的两个核心点

> 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。

vue和jQuery的区别

> 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。

SPA首屏加载慢如何解决

> 安装动态懒加载所需插件；使用CDN资源。

如何优化SPA应用的首屏加载速度慢的问题

> 将公用的JS库通过script标签外部引入，减小app.bundel的大小，让浏览器并行下载资源文件，提高下载速度；
>
> 在配置 路由时，页面和组件使用懒加载的方式引入，进一步缩小 app.bundel 的体积，在调用某个组件时再加载对应的js文件；
>
> 加一个首屏 loading 图，提升用户体验；

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。

`vue2` 和 `vue3` 有何区别,`vue3` 解决了哪些 `vue2` 无法解决的问题

“

vue与Angular react的区别

“
1.与AngularJS的区别

相同点：

都支持指令：内置指令和自定义指令；都支持过滤器：内置过滤器和自定义过滤器；都支持双向数据绑定；都不支持低端浏览器。

不同点：

AngularJS的学习成本高，比如增加了Dependency Injection特性，而Vue.js本身提供的API都比较简单、直观；在性能上，AngularJS依赖对数据做脏检查，所以Watcher越多越慢；Vue.js使用基于依赖追踪的观察并且使用异步队列更新，所有的数据都是独立触发的。

2.与React的区别

相同点：

React采用特殊的JSX语法，Vue.js在组件开发中也推崇编写.vue特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用；中心思想相同：一切都是组件，组件实例之间可以嵌套；都提供合理的钩子函数，可以让开发者定制化地去处理需求；都不内置列数AJAX，Route等功能到核心包，而是以插件的方式加载；在组件开发中都支持mixins的特性。

不同点：

React采用的Virtual DOM会对渲染出来的结果做脏检查；Vue.js在模板中提供了指令，过滤器等，可以非常方便，快捷地操作Virtual DOM。

服务端渲染(英文缩写SSR)的好处

> “

vue为什么不兼容ie8版本及以下的浏览器

“

多角色登录时、如何处理用户权限的问题(开放题)

“

web

浏览器中的事件循环机制（Event Looping）

“
JS在执行的过程中会产生一个执行环境，js的代码在这个执行环境中按顺序执行，当遇到异步代码时，会被挂起到事件队列中（Task），对于这个队列中的任务则会涉及到微任务和宏任务，当执行栈的为空时，由于事件循环机制，就会从事件队列中拿出需要执行的代码到执行栈中执行，此时执行的循序时先执行微任务，当微任务执行完毕之后，再执行宏任务。

前端工程化

“
早期的非工程化前端开发方式，与小作坊相似:

按照个人习惯制作 html 页面 使用 jQuery 等技术添加一些动态效果与数据 随便找个 框架 改一改

总之：没有一个固定的规矩可以遵循，没有标准化的操作流程．很难保证质量．

前端工程化就是形成一套规矩，把前端网站的制作标准化，大概分为以下措施：

模块化

把耦合在一起的大文件 拆分成功能独立的小文件．再进行统一的拼装和加载．这样才能多人协作．例如 JS 的模块化操作：commonJS ，AMD ，CMD webpack 工具：进行模块的打包

组件化 代码的设计层面，把不同的功能解耦合，设计成可插拔的组件．相当于：台式机与笔记本的差别．台式机的各个零件都可以随意替换 而 不会影响其他组件 规范化 设定一个规范，让所有参与人员的代码统一风格，便于团队协作与维护．

目录结构的制定 代码规范 前后端接口规范 文档规范 组件管理 git分支管理 commit 描述规范 定期 Code Review 视觉图标规范

自动化

任何简单机械的重复劳动 都应该让机器自动完

图标合并:webpack -- 雪碧图

自动化构建：脚手架

自动化部署：脚手架

自动化测试：脚手架

cookies，sessionStorage和localStorage的区别

> 共同点：都是保存在浏览器端，且是同源的。

区别：

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

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

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

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

前端性能优化

> https://www.cnblogs.com/qianguyihao/p/8550195.html
>
> 提升页面性能优化的常见方式：
>
> 1、资源压缩合并，减少http请求
>
> 2、非核心代码异步加载 --> 异步加载的方式 --> 异步加载的区别
>
> 3、利用浏览器缓存 --> 缓存的分类 --> 缓存的原理
>
> 缓存是所有性能优化的方式中最重要的一步【重要】

4、使用CDN

浏览器第一次打开页面时，缓存是起不了作用的。这个时候，CDN就上场了。

5、DNS预解析

兼容性问题如何解决

token的签发（jwt）

UI给了一个750宽度的设计图纸，如何适应不同终端；

前端性能优化方案

> 1、减少 DOM 操作 2、部署前，图片压缩，代码压缩 3、优化 js 代码结构，减少
>
> 冗余代码 4、减少 http 请求，合理设置 HTTP 缓存 5、使用内容分发 cdn 加速 6、
>
> 静态资源缓存 7、图片延迟加载

网页从输入网址到渲染完成经历了哪些过程

> 1. 输入网址；
> 2. 发送到DNS服务器，并获取域名对应的web服务器对应的ip地址；
> 3. 与web服务器建立TCP连接；
> 4. 浏览器向web服务器发送http请求；
> 5. web服务器响应请求，并返回指定url的数据（或错误信息，或重定向的新的url地址）；
> 6. 浏览器下载web服务器返回的数据及解析html源文件；

> 生成DOM树，解析css和js，渲染页面，直至显示完成；

H5有哪些本地数据存储方式分别适用于什么情景

> “

Rem布局的原理是什么如何解决Rem换算PX时页面先缩小后放大的问题

> “

如何实现浏览器内多个标签叶之间的通信

> “

经常遇到的浏览器兼容性有哪些原因,解决方法是什么,常用hack的技巧

> “

页面怎么扁平化

给你一个电池,里面动态显示电量,用动态效果做出来

js加载机制

简述xss,怎么避免

写出几种IE6 BUG的解决方法

> 1.双边距BUG float引起的 使用display
> 2.3像素问题 使用float引起的 使用dislpay:inline -3px
> 3.超链接hover 点击后失效 使用正确的书写顺序 link visited hover active
> 4.Ie z-index问题 给父级添加position:relative
> 5.Png 透明 使用js代码 改
> 6.Min-height 最小高度 !Important 解决’
> 7.select 在ie6下遮盖 使用iframe嵌套
> 8.为什么没有办法定义1px左右的宽度容器(IE6默认的行高造成的，使用over:hidden,zoom:0.08 line-height:1px)
> 9.ie 6 不支持!important

什么叫优雅降级和渐进增强

> 渐进增强 progressive enhancement：
>
> 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、交互等改进和追加功能达到更好的用户体验。
>
> 优雅降级 graceful degradation：
>
> 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。
>
> 区别：
>
> a. 优雅降级是从复杂的现状开始，并试图减少用户体验的供给
>
> b. 渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要

> c. 降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

浏览器的内核分别是什么

> IE：trident内核
>
> Firefox：gecko内核
>
> Safari：webkit内核
>
> Opera：以前是presto内核，Opera现已改用Google Chrome的Blink内核
>
> Chrome：Blink(基于webkit，Google与Opera Software共同开发)

对前端工程师这个职位你是怎么样理解的

> a. 前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100 分，甚至更好
>
> b. 参与项目，快速高质量完成实现效果图，精确到1px；
>
> c. 与团队成员，UI设计，产品经理的沟通；
>
> d. 做好的页面结构，页面重构和用户体验；
>
> e. 处理hack，兼容、写出优美的代码格式；
>
> f. 针对服务器的优化、拥抱最新前端技术。

你如何优化自己的代码

> 1.代码重用2.避免全局变量（命名空间，封闭空间，模块化mvc..）
>
> 3.拆分函数避免函数过于臃肿4.注释

如何对Web系统进行性能优化

> 1）减少一个页面访问所产生的http连接次数；
>
> 2）尽量简洁的页面设计，最大程度减少图片的使用，通过放弃一些不必要的页面特效来减少javascript的使用；
>
> 3）使用一些优化技巧，比如利用图片的背景位移减少图片的个数；image map技术；使用Inline images将css图片捆绑到网页中；
>
> 4）尽量合并js和css文件，减少独立文件个数；
>
> 5）使用gzip来压缩网页中的静态内容，能够显著减少用户访问网页时的等待时间；
>
> 6）CSS的引用要放在html的头部header中，JS文件引用尽量放在页面底端标签的后面，主要的思路是让核心的页面内容尽早显示出来；
>
> 7）不要在网页中引用太多的外部脚本；
>
> 8）避免重定向的发生；

1、如何根据需求，自己独立搭建项目，或者说搭建项目的思路是怎样的

> 搭建项目首先要选择合适的规范来写代码，可以采用es6的模块化规范来写代码，另外再选择合适的构建工具，确定是单页面应用还是多页面应用，然后再选择合适的前端框架，定好项目的目录结构，然后再搭建一个好的脚手架，项目搭建好之后再使用版本控制系统来管理源代码，之后就开始编写代码

一个网页JS内存消耗大,怎么优化

> 进行 js 封装，尽量复用代码.
>
> 尽量减少闭包的使用
>
> 不要定义全局变量（1.使变量不易被回收；2.多人协作时容易产生混淆；3.在作用域中 容易被干扰）
>
> js 避免嵌套循环

页面自适应如何处理

> 1.采用 bootstrap 的的网格系统来进行页面布局划分
>
> 2.整个页面的宽度 width 和高度 height 都采用百分比的形式来设计
>
> 3.字体如何去做自适应我采用的是使用 Css3 的 @media 查询，通过@media 可以针对
>
> 不同的屏幕尺寸设置不同的样式

简述vue、angular、react各自使用场景

> vue因为上手快、语法简洁 可以被快速的使用在中小型项目中
>
> anguar模块多，学习曲线陡峭，更多的用在中大型超大型的项目中
>
> react被用在实现一些对于DOM操作非常频繁的场景里

react

请说下什么叫redux和redux-saga

对柯里化的理解

ReactJS中渲染根组件的方式以及注意事项

> ReactDOM.render(A,B);将A渲染到B指定的容器中
>
> 注意事项:
>
> 不允许一次渲染多个标签，但是可以放在同一个顶层标签
>
> 每一个标记都要有对应的结束

ReactJS中父子组件通信的方式

> (1) 父与子通信

> 借助于属性 往下传值
>
> 传值:
>
> 接受值:
>
> this.props.myName
>
> (2) 子与父通信
>
> 通过属性传递有参数的方法 让子组件调用是传值
>
> ①定义有参数的方法
>
> rcvMsg(msg){}
>
> ②传递给子组件
>
> ③子组件来调用
>
> This.props.funcRcv(123)

如何在组件渲染时，调用组件内部嵌套的子组件

> This.props.children

组件的生命周期

> mount:
>
> componentWillMount
>
> componentDidMount
>
> update:
>
> componentWillUpdate

componentDidUpdate

componentWillReceiveProps

unmount:

componentWillUnmount

react中是如何处理网络通信的

> fetch(url).then((response)=>response.json()).then((result)=>{})

react中循环创建多个组件时指定key的作用

> 在dom变化时 快速定位元素 提升更新效率

react的生态圈(技术栈)中有哪些常见技术

> reactjs、 reactNative、 react360 、flux、 redux、 ssr、 reactNavigation....

基于reactNative的reactNavigation中的基础用法

> 跳转:
>
> this.props.navigation.navigate()
>
> 传参:
>
> this.props.navigation.navigate('detail',{id:10})

> this.props.navigation.getParam('id')

React的一些主要优点
React有哪些限制
React 中 render() 的目的
React中的状态是什么它是如何使用的
React组件生命周期的阶段是什么
如何模块化 React 中的代码
数据如何通过 Redux 流动
Redux与Flux有何不同
为什么需要 React 中的路由

angular

angular 双向绑定原

> angular.js是通过脏值检测的方式，对比数据是否有变更，从而决定是否更新视图。简单的方式就是通过setInterval()定时轮询检测数据变动。angular.js只有在指定的事件触发时，进入脏值检测，大致如下： - DOM事件，譬如用户输入文本，点击按钮等（ng-click） - XHR响应事件（$http） - 浏览器location变更事件（$location） - Timer事件（$timeout,$interval） - 执行$digest()或$apply()

Angular中组件之间通信的方式

> Props down
>
> 1、调用子组件时 通过自定义属性传值
>
> 2、子组件内部通过Input来接受属性的值
>
> Events up
>
> \1. 在父组件中定义一个有参数的方法
>
> \2. 调用子组件时，绑定自定义事件和上一步的方法
>
> \3. 子组件内部通过Output和EventEmitter来触发事件并传值

Angular的八大组成部分并简单描述

> Module 是Angular开发中的基本单位，是一个容器，可以包含组件、指令、管道等
>
> Components 是可被反复使用的 带有特定功能的视图
>
> Templates 是经过指令和管道、组件等增强过的html
>
> Bindings 结合着事件绑定 属性绑定 双向数据绑定等扩展html的功能
>
> Directives 分为结构性和属性型指令还有其他模块中比如路由模块中的指令等，主要是增强html
>
> Pipes 可以对数据做一个筛选、过滤、格式化从而得到目的数据
>
> Service 将组件、应用中的可共用的部分 ，比如数据或者方法等 封装成服务以方便服用

> DependencyInjection　依赖注入

Angular中常见的生命周期的钩子函数

> ngOnChanges：当Angular设置其接收当前和上一个对象值的数据绑定属性时响应。
>
> ngOnInit：在第一个ngOnChange触发器之后，初始化组件/指令。这是最常用的方法，用于从后端服务检索模板的数据。
>
> ngDoCheck：检测并在Angular上下文发生变化时执行。每次更改检测运行时，会被调用。
>
> ngOnDestroy：在Angular销毁指令/组件之前清除。取消订阅可观察的对象并脱离事件处理程序，以避免内存泄漏。
>
> 组件特定hooks：
>
> ngAfterContentInit：组件内容已初始化完成
>
> ngAfterContentChecked：在Angular检查投影到其视图中的绑定的外部内容之后。
>
> ngAfterViewInit：Angular创建组件的视图后。
>
> ngAfterViewChecked：在Angular检查组件视图的绑定之后。

Angular中路由的工作原理

> Angular应用程序具有路由器服务的单个实例，并且每当URL改变时，相应的路由就与路由配置数组进行匹配。在成功匹配时，它会应用重定向，此时路由器会构建ActivatedRoute对象的树，同时包含路由器的

当前状态。在重定向之前，路由器将通过运行保护（CanActivate）来检查是否允许新的状态。Route Guard只是路由器运行来检查路由授权的接口方法。保护运行后，它将解析路由数据并通过将所需的组件实例化到 </ router-outlet>中来激活路由器状态。

解释rxjs在Angular中的使用场景

> Rxjs是微软所提供的一种的异步处理数据的方式，在Angular中处理网络通信时用到了。
> 创建一个Observable并subsribe
>
> 比如: this.http.get('').subscribe((data)=>{})

ng中*ngFor和*ngIf的使用注意事项

> 不能在一个元素上同时使用结构行指令，同时使用可以通过ng-container来避免报错

描述下angular中路由的背后原理(spa的工作方式)

> 监控地址栏的变化
>
> 解析地址栏中的路由地址
>
> 将路由地址和路由数组中的每个路有对象的path属性进行匹配，如果匹配上就会加载
>
> 的视图内容到routerOutlet中进行显示

微信小程序

微信小程序不同页面如何传值；

小程序的app和page的生命周期，触发顺序是什么；

\1. 小程序和移动端使用场景

(1) 小程序默认弹窗

微信小程序组件的生命周期

> 生命周期函数-onLoad：页面加载
>
> 一个页面只会调用一次，可以在 onLoad 中获取打开当前页面所调用的 query 参数。
>
> -onShow：页面显示
>
> 每次打开页面都会调用一次。
>
> -onReady：页面初次渲染完成
>
> 一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。
>
> -onHide：页面隐藏
>
> 当navigateTo或底部tab切换时调用。
>
> -onUnload：页面卸载

简单描述下微信小程序的相关文件类型

> 微信小程序项目结构主要有四个文件类型,如下
>
> .json 后缀的 JSON 配置文件
>
> .wxml 后缀的 WXML 模板文件
>
> .wxss 后缀的 WXSS 样式文件
>
> .js 后缀的 JS 脚本逻辑文件

小程序onPageScroll方法的使用注意什么**

> 由于此方法调用频繁，不需要时，可以去掉，不要保留空方法，并且使用onPageScroll时，尽量避免使用setData()，尽量减少setData()的使用频次。

end