《前端初级工程师面试题精编解析大全》

前言

历时半年，我们整理了这份市面上最全面的前端初级工程师面试题解析大全。

包含了腾讯、百度、小米、阿里、乐视、美团、58、猎豹、360、新浪、搜狐等一线互联网公司面试被问到的题目。熟悉本文中列出的知识点会大大增加通过前两轮技术面试的几率。

特意推荐给前端开发的朋友们，定价友好只需要 29.9 元。请前往支持正版。

前端面试题集锦——HTML 篇

1. 你是怎么理解 HTML 语义化

Step 1：先举例说明
HTML 语义化简单来说就是用正确的标签来做正确的事。
比如表示段落用 p 标签、表示标题用 h1-h6 标签、表示文章就用 article 等。
Step 2：说说为什么需要使用语义化标签
前期：前端工作主要由后端人员完成，也就是打野阶段，使用的是 table 布局
中期：美工人员使用 div+css 来完成
当前：专业的前端开发应该使用合适的标签来表达正确含义的页面结构
让页面具有良好的结构和含义，可以有效提高：
可访问性：帮助辅助技术更好的阅读和转译你的网页，利于无障碍阅读；
可检索性：有了良好的结构和语义，可以提高搜索引擎的有效爬取，提高网站流量；
国际化：全球只有 13%的人口是英语母语使用者，因此通用的语义化标签可以让各国开发者更容易弄懂你网页的结构；

互用性：减少网页间的差异性，帮助其他开发者了解你网页的结构，方便后期开发和维护

2. 你用过哪些 HTML5 标签

表示结构的标签

<header> <nav> <main> <article> <section> <aside> <footer>

表示文字形式

<data>：

举例：展示了一些产品名称，而且每个名称都和一个产品编码相关联。

新产品 <ul> <data value="398">迷你番茄酱</data> <data value="399">巨无霸番茄酱</data> <data value="400">超级巨无霸番茄酱</data>

<time>：

表示日期和时间值，机器读取通过 datetime 属性指定。

举例：

The concert took place on <time datetime="2001-05-15 19:00">May 15</time>.

<mark>：用于高亮文本

嵌入内容

<video>：

controls：展示视频自带的控件

autoplay：视频马上自动播放

poster：海报帧的 URL

height、width：视频显示区域的宽和高

loop：视频结尾自动回到视频开始的地方

<video controls> <source src="myVideo.mp4" type="video/mp4"> <source src="myVideo.webm" type="video/webm"> Your browser doesn't support HTML5 video. Here is a link to the video instead. </video><!-- Simple video example --><video src="videofile.ogg" autoplay poster="posterimage.jpg"> 抱歉，您的浏览器不支持内嵌视频，不过不用担心，你可以 下载 并用你喜欢的播放器观看 !</video><!-- Video with subtitles --><video src="foo.ogg"> <track kind="subtitles" src="foo.en.vtt" srclang="en" label="English"> <track kind="subtitles" src="foo.sv.vtt" srclang="sv" label="Svenska"></video>

<audio>：

大部分同 <vedio>

controls：展示音频自带的控件

autoplay：音频马上自动播放

muted：是否静音

loop：音频结尾自动回到开始的地方

<audio controls> <source src="myAudio.mp3" type="audio/mpeg"> <source src="myAudio.ogg" type="audio/ogg"> Your browser doesn't support HTML5 audio. Here is a link to the audio instead. </audio><!-- Simple audio playback --><audio src="AudioTest.ogg" autoplay> Your browser does not support the <code>audio</code> element.</audio><audio controls> <source src="foo.opus"

type="audio/ogg; codecs=opus"/> <source src="foo.ogg" type="audio/ogg; codecs=vorbis"/> <source src="foo.mp3" type="audio/mpeg"/></audio>复制代码

<canvas> ：

通过 JavaScript 和 HTML 的 <canvas> 元素来绘制图形

<canvas id="canvas" width="300" height="300"></canvas>复制代码

//获取 HTML <canvas> 元素的引用 const canvas = document.getElementById('canvas');//获得一个绘图上下文 const ctx = canvas.getContext('2d');//让长方形变成绿色 ctx.fillStyle = 'green';//将它的左上角放在(10, 10)，把它的大小设置成宽 150 高 100ctx.fillRect(10, 10, 150, 100);

3. meta viewport 是做什么用的，怎么写？

Step 1：使用目的

是为了在移动端不让用户缩放页面使用的

Step 2：怎么写

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale-1, minimum-scale=1">

Step 3：解释每个单词的含义

with=device-width 将布局视窗（layout viewport）的宽度设置为设备屏幕分辨率的宽度

initial-scale=1 页面初始缩放比例为屏幕分辨率的宽度

maximum-scale=1 指定用户能够放大的最大比例

minimum-scale=1 指定用户能够缩小的最大比例

4.H5 是什么

简单粗暴：就是一种移动端页面

深入点：微信上的一种移动营销页面

总之不是 HTML5

5.label 标签的作用

label 标签来定义表单控制间的关系,当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。

<label for="Name">Number:</label><input type='text' name="Name" id="Name"/><label>Date:<input type="text" name="B"/></label>

6.行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？

首先：CSS 规范规定，每个元素都有 display 属性，确定该元素的类型，每个元素都有默认的 display 值，如 div 的 display 默认值为“block”，则为“块级”元素；span 默认 display 属性值为“inline”，是“行内”元素。
常用的块状元素有： 、<p>、<h1>...<h6>、<ol>、<ul>、<dl>、<table>、<address>、<blockquote> 、<form>
常用的内联元素有： <a>、、
、<i>、、、<label>、<q>、<var>、<cite>、<code>
常用的内联块状元素有： <img>、<input>
知名的空元素：
 <hr/> <img/> <input/> <link/> <meta/>

7.a 标签中 如何禁用 href 跳转页面 或 定位链接

e.preventDefault();href="javascript:void(0);

8. canvas 在标签上设置宽高 和在 style 中设置宽高有什么区别

canvas 标签的 width 和 height 是画布实际宽度和高度，绘制的图形都是在这个上面。而 style 的 width 和 height 是 canvas 在浏览器中被渲染的高度和宽度。如果 canvas 的 width 和 height 没指定或值不正确，就被设置成默认值 。

9.你做的页面在哪些流览器测试过？这些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 GoogleChrome 的 Blink 内核

Chrome:Blink(基于 webkit，Google 与 Opera Software 共同开发)

10.iframe 有哪些缺点？

iframe 是一种框架，也是一种很常见的网页嵌入方
iframe 的优点：
1.iframe 能够原封不动的把嵌入的网页展现出来。
2.如果有多个网页引用 iframe，那么你只需要修改 iframe 的内容，就可以实现调用的每一个页面内容的更改，方便快捷。
3.网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用 iframe 来嵌套，可以增加代码的可重用。
4.如果遇到加载缓慢的第三方内容如图标和广告，这些问题可以由 iframe 来解决。
iframe 的缺点：
1.会产生很多页面，不容易管理。
2.iframe 框架结构有时会让人感到迷惑，如果框架个数多的话，可能会出现上下、左右滚动条，会分散访问者的注意力，用户体验度差。
3.代码复杂，无法被一些搜索引擎索引到，这一点很关键，现在的搜索引擎爬虫还不能很好的处理 iframe 中的内容，所以使用 iframe 会不利于搜索引擎优化。
4.很多的移动设备（PDA 手机）无法完全显示框架，设备兼容性差。
5.iframe 框架页面会增加服务器的 http 请求，对于大型网站是不可取的。
现在基本上都是用 Ajax 来代替 iframe，所以 iframe 已经渐渐的退出了前端开发。

11.HTML5 新特性

1. 本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失; sessionStorage 的数据在浏览器关闭后自动删除;
新的技术 webworker, websocket, Geolocation;

12.HTML5 离线储存

在用户没有与因特网连接时，可以正常访问站点或应用，在用户与因特网连接时，更新用户机器上的缓存文件。
原理： HTML5 的离线存储是基于一个新建的.appcache 文件的缓存机制(不是存储技术)，通过这个文件上的解析清单离线存储资源，这些资源就会像 cookie 一样被存储了下来。之后当网络在处于离线状态下时，浏览器会通过被离线存储的数据进行页面展示。
如何使用：
页面头部像下面一样加入一个 manifest 的属性；

在 cache.manifest 文件的编写离线存储的资源：

```
CACHE MANIFEST
#v0.11
CACHE:
js/app.js
css/style.css
NETWORK:
resourse/logo.png
FALLBACK:
/ /offline.html
```

在离线状态时，操作 window.applicationCache 进行需求实现。

13.浏览器是怎么对 HTML5 的离线储存资源进行管理和加载的呢？

在线的情况下，浏览器发现 html 头部有 manifest 属性，它会请求 manifest 文件，如果是第一次访问 app，那么浏览器就会根据 manifest 文件的内容下载相应的资源并且进行离线存储。如果已经访问过 app 并且资源已经离线存储了，那么浏览器就会使用离线的资源加载页面，然后浏览器会对比新的 manifest 文件与旧的 manifest 文件，如果文件没有发生改变，就不做任何操作，如果文件改变了，那么就会重新下载文件中的资源并进行离线存储。
离线的情况下，浏览器就直接使用离线存储的资源。

14.Doctype 作用？严格模式与混杂模式如何区分？它们有何意义？

（1）、<!DOCTYPE> 声明位于文档中的最前面，处于 <html> 标签之前。告知浏览器以何种模式来渲染文档。
（2）、严格模式的排版和 JS 运作模式是 以该浏览器支持的最高标准运行。
（3）、在混杂模式中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。
（4）、DOCTYPE 不存在或格式不正确会导致文档以混杂模式呈现。复制代码
你知道多少种 Doctype 文档类型？
该标签可声明三种 DTD 类型，分别表示严格版本、过渡版本以及基于框架的 HTML 文档。
HTML 4.01 规定了三种文档类型：Strict、Transitional 以及 Frameset。
XHTML 1.0 规定了三种 XML 文档类型：Strict、Transitional 以及 Frameset。
Standards （标准）模式（也就是严格呈现模式）用于呈现遵循最新标准的网页，而 Quirks （包容）模式（也就是松散呈现模式或者兼容模式）用于呈现为传统浏览器而设计的网页。

15.HTML 与 XHTML——二者有什么区别

区别：
1.所有的标记都必须要有一个相应的结束标记
2.所有标签的元素和属性的名字都必须使用小写
3.所有的 XML 标记都必须合理嵌套
4.所有的属性必须用引号""括起来
5.把所有<和&特殊符号用编码表示
6.给所有属性赋一个值
7.不要在注释内容中使“--”
8.图片必须有说明文字复制代码

前端面试题集锦——CSS 篇

1. 页面渲染时，dom 元素所采用的 布局模型,可通过 box-sizing 进行设置。根据计算宽高的区域可分为：

content-box (W3C 标准盒模型) border-box (IE 盒模型) padding-box (FireFox 曾经支持) margin-box (浏览器未实现)

Tips: 理论上是有上面 4 种盒子，但现在 w3c 与 mdn 规范中均只支持 content-box 与 border-box；

2. ie 盒模型算上 border、padding 及自身（不算 margin），标准的只算上自身窗体的大小 css 设置方法如下：

标准模型 :box-sizing:content-box; IE 模型:box-sizing:border-box;复制代码

3.几种获得宽高的方式 ：

dom.style.width/height
这种方式只能取到 dom 元素内联样式所设置的宽高，也就是说如果该节点的样式是在 style 标签中或外联的 CSS 文件中设置的话，通过这种方法是获取不到 dom 的宽高的。

dom.currentStyle.width/height
这种方式获取的是在页面渲染完成后的结果，就是说不管是哪种方式设置的样式，都能获取到。但这种方式只有 IE 浏览器支持。
window.getComputedStyle(dom).width/height
这种方式的原理和 2 是一样的，这个可以兼容更多的浏览器，通用性好一些。
dom.getBoundingClientRect().width/height
这种方式是根据元素在视窗中的绝对位置来获取宽高的
dom.offsetWidth/offsetHeight
这个就没什么好说的了，最常用的，也是兼容最好的。

4.拓展各种获得宽高的方式 ：

获取屏幕的高度和宽度（屏幕分辨率）： window.screen.height/width
获取屏幕工作区域的高度和宽度（去掉状态栏）： window.screen.availHeight/availWidth
网页全文的高度和宽度： document.body.scrollHeight/Width
滚动条卷上去的高度和向右卷的宽度： document.body.scrollTop/scrollLeft
网页可见区域的高度和宽度（不加边线）： document.body.clientHeight/clientWidth 网页可见区域的高度和宽度（加边线）： document.body.offsetHeight/offsetWidth

5.边距重叠解决方案(BFC) BFC 原理 ：

内部的 box 会在垂直方向，一个接一个的放置 每个元素的 margin box 的左边，与包含块 border box 的左边相接触（对于从做往右的格式化，否则相反）
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
bfc 的区域不会与浮动区域的 box 重叠
bfc 是一个页面上的独立的容器，外面的元素不会影响 bfc 里的元素，反过来，里面的也不会影响外面的
计算 bfc 高度的时候，浮动元素也会参与计算 创建 bfc float 属性不为 none（脱离文档流）
position 为 absolute 或 fixed
display 为 inline-block,table-cell,table-caption,flex,inine-flex
overflow 不为 visible 根元素 demo
上
这块 margin-bottom:30px;
下
这块 margin-top:50px;

6.css reset 和 normalize.css 有什么区别：

两者都是通过重置样式，保持浏览器样式的一致性
前者几乎为所有标签添加了样式，后者保持了许多浏览器样式，保持尽可能的一致
后者修复了常见的桌面端和移动端浏览器的 bug：包含了 HTML5 元素的显示设置、预格式化文字的 font-size 问题、在 IE9 中 SVG 的溢出、许多出现在各浏览器和操作系统中的与表单相关的 bug。
前者中含有大段的继承链
后者模块化，文档较前者来说丰富

7.居中方法：

1. 水平方向上
针对 inline，内联块 inline-block，内联表 inline-table, inline-flex 元素及 img,span,button 等元素.text_div{
text-align:center;
}
不定宽块状元素居中
.text_div{
margin:0 auto;//且需要设置父级宽度
}
通过给父元素设置 float，然后给父元素设置 position:relative 和 left:50%，子元素设置 position:relative 和 left: -50% 来实现水平居中。.wrap{

float:left; position:relative; left:50%; clear:both; }.wrap-center{ left:-50%;
}
2. 垂直居中
单行内联(inline-)元素垂直居中
通过设置内联元素的高度(height)和行高(line-height)相等，从而使元素垂直居中。
.text_div{
height: 120px; line-height: 120px; }
利用表布局.father { display: table; }.children {
display: table-cell; vertical-align: middle; text-align: center;
}
flex 布局
.center-flex { display: flex;
flex-direction: column;//上下排列
justify-content: center; }
绝对布局方式
已知高度.parent { position: relative; }.child {
position: absolute;
top: 50%; height: 100px;
margin-top: -50px; }
未知高度.parent { position: relative; }.child {
position: absolute;

top: 50%;
transform: translateY(-50%); }
3. 垂直水平居中根据上方结合
flex 方式.parent { display: flex; justify-content: center; align-items: center; }grid 方式.parent { height: 140px;
display: grid; }.child { margin: auto; }

8.css 优先确定级：

每个选择器都有权值，权值越大越优先
继承的样式优先级低于自身指定样式
！important 优先级最高 js 也无法修改
权值相同时，靠近元素的样式优先级高 顺序为内联样式表（标签内部）> 内部样式表（当前文件中）> 外部样式表（外部文件中）

9.如何清除浮动：

不清楚浮动会发生高度塌陷：浮动元素父元素高度自适应（父元素不写高度时，子元素写了浮动后，父元素会发生高
度塌陷）

clear 清除浮动（添加空 div 法）在浮动元素下方添加空 div,并给该元素写 css 样式：{clear:both;height:0;overflow:hidden;}
给浮动元素父级设置高度
父级同时浮动（需要给父级同级元素添加浮动）
父级设置成 inline-block，其 margin: 0 auto 居中方式失效给父级添加 overflow:hidden 清除浮动方法
万能清除法 after 伪类 清浮动（现在主流方法，推荐使用）
float_div:after{ content:"."; clear:both; display:block; height:0; overflow:hidden; visibility:hidden; }.float_div{ zoom:1
}
.clearfix:after {
/生成内容作为最后一个元素/
content: "";
/使生成的元素以块级元素显示,占满剩余空间/
display: block;
/避免生成内容破坏原有布局的高度/
height: 0;
/使生成的内容不可见，并允许可能被生成内容盖住的内容可以进行点击和交互/
visibility: hidden;
/清除浮动元素对当前元素的影响/

```
clear: both; }
.clearfix {
/用于兼容 IE，触发 IE hasLayout/
*zoom:1;
}
```

10.自适应布局：

左侧浮动或者绝对定位，然后右侧 margin 撑开使用 div 包含，然后靠负 margin 形成 bfc
使用 flex

11.画三角形：

```
#item { width: 0; height: 0;
border-left: 50px solid transparent; border-right: 50px solid transparent; border-top: 50px solid transparent; border-bottom: 50px solid blue; background: white;
}
```

12.link @import 导入 css：

link 是 XHTML 标签，除了加载 CSS 外，还可以定义 RSS 等其他事务；@import 属于 CSS 范畴，只能加载 CSS。
link 引用 CSS 时，在页面载入时同时加载；@import 需要页面网页完全载入以后加载。link 无兼容问题；@import 是在 CSS2.1 提出的，低版本的浏览器不支持。
link 支持使用 Javascript 控制 DOM 去改变样式；而@import 不支持。

13.长宽比方案：

使用 padding 方式结合 calc 实现
长宽一项设置百分比另一项 aspect-ratio 实现（需借助插件实现）

14.display 相关：

block:div 等容器类型 inline:img span 等行内类型
table 系列：将样式变成 table 类型 flex:重点把握，非常强大
grid:同上
inline-block:可设置宽度，两者间有一点间隙 inherit:继承父级

15.CSS 优化：

层级扁平，避免过于多层级的选择器嵌套；
特定的选择器 好过一层一层查找: .xxx-child-text{} 优于 .xxx .child .text{}减少使用通配符与属性选择器；
减少不必要的多余属性；
使用 动画属性 实现动画，动画时脱离文档流，开启硬件加速，优先使用 css 动画；使用 替代原生 @import；

16.CSS 开启 GPU 加速

为动画 DOM 元素添加 CSS3 样式 -webkit-transform:transition3d(0,0,0) 或 -webkit-transform:translateZ(0);，这两个属性都会开启 GPU 硬件加速模式，从而让浏览器在渲染动画时从 CPU 转向 GPU，其实说白了这是一个小伎俩，也可以算是一个 Hack，-webkit-transform:transition3d 和-webkit- transform:translateZ 其实是为了渲染 3D 样式，但我们设置值为 0 后，并没有真正使用 3D 效果，但浏览器却因此开启了 GPU 硬件加速模式。

17.开启 GPU 硬件加速可能触发的问题：

通过-webkit-transform:transition3d/translateZ 开启 GPU 硬件加速之后，有些时候可能会导致浏览器频繁闪烁或抖动，可以尝试以下办法解决之：
-webkit-backface-visibility:hidden;
-webkit-perspective:1000;复制代码

18.CSS 中 link 与@import 的区别：

@import 是 CSS 提供的语法规则，只有导入样式表的作用；link 是 HTML 提供的标签，不仅

可以加载 CSS 文件，还可以定义 RSS、rel 连接属性等。
加载页面时，link 引入的 CSS 被同时加载，@import 引入的 CSS 将在页面加载完毕后加载。
link 标签作为 HTML 元素，不存在兼容性问题，而@import 是 CSS2.1 才有的语法，故老版本浏览器（IE5 之前）不能识别。
可以通过 JS 操作 DOM，来插入 link 标签改变样式；由于 DOM 方法是基于文档的，无法使用@import 方式插入样式。

19.CSS 选择器列表优先级及权重：

通用选择器（*）
元素(类型)选择器 权重 1 类选择器 权重 10
属性选择器
伪类
ID 选择器 权重 100
内联样式 权重 1000 !important 规则会覆盖任何其他的声明，只在需要覆盖全站或外部 CSS 的替丁页面中使用。

20.display:none 和 visibility:hidden 的区别：

display:none ： 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。
visibility:hidden ： 隐藏对应的元素，但是在文档布局中仍保留原来的空间。复制代码

21.position 的 absolute 与 fixed 共同点与不同点：

共同点：
1.改变行内元素的呈现方式，display 被置为 block；
2.让元素脱离普通流，不占据空间；
3.默认会覆盖到非定位元素上不同点：
absolute 的”根元素“是可以设置的，而 fixed 的”根元素“固定为浏览器窗口。当你滚动网页，fixed 元素与浏览器窗口之间的距离是不变的。复制代码

22.介绍一下 CSS 的盒子模型：

有两种， IE 盒子模型、标准 W3C 盒子模型；IE 的 content 部分包含了 border 和 pading;

盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border).复制代码

23.CSS 选择符有哪些？

id 选择器（ # myid）
类选择器（.myclassname）
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（ * ）
属性选择器（a[rel = "external"]）
伪类选择器（a: hover, li:nth-child）

24.哪些属性可以继承？

可继承的样式： font-size font-family color, text-indent;不可继承的样式：border padding margin width height ;

25.优先级算法如何计算？

优先级就近原则，同权重情况下样式定义最近者为准
载入样式以最后载入的定位为准;优先级为:
!important > id > class > tag
important 比 内联优先级高,但内联比 id 要高

26.CSS3 新增伪类有哪些：

CSS3 新增伪类举例：
p:first-of-type 选择属于其父元素的首个元素的每个
元素。
p:last-of-type 选择属于其父元素的最后元素的每个
元素。
p:only-of-type 选择属于其父元素唯一的元素的每个
元素。
p:only-child 选择属于其父元素的唯一子元素的每个元素。
p:nth-child(2) 选择属于其父元素的第二个子元素的每个元素。
:enabled :disabled 控制表单控件的禁用状态。:checked 单选框或复选框被选中。

27.列出 display 的值，说明他们的作用：

block 象块类型元素一样显示。
inline 缺省值。象行内元素类型一样显示。
inline-block 象行内元素一样显示，但其内容象块类型元素一样显示。list-item 象块类型元素一样显示，并添加样式列表标记。

28.position 的值，relative 和 absolute 分别是相对于谁进行定位的：

absolute
生成绝对定位的元素，相对于 static 定位以外的第一个祖先元素进行定位。
fixed （老 IE 不支持）
生成绝对定位的元素，相对于浏览器窗口进行定位。
relative
生成相对定位的元素，相对于其在普通流中的位置进行定位。
static 默认值。没有定位，元素出现在正常的流中*（忽略 top, bottom, left, right z-index 声明）。inherit 规定从父元素继承 position 属性的值。

29. CSS3 有哪些新特性：

CSS3 实现圆角（border-radius），阴影（box-shadow），
对文字加特效（text-shadow、），线性渐变（gradient），旋转（transform）
transform:rotate(9deg) scale(0.85,0.90) translate(0px,-30px) skew(-9deg,0deg);//旋转,缩放,定位,倾斜增加了更多的 CSS 选择器 多背景 rgba
在 CSS3 中唯一引入的伪元素是::selection.媒体查询，多栏布局
border-image 复制代码

30.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。
当然，初始化样式会对 SEO 有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况

下初始化。
*最简单的初始化方法就是： * {padding: 0; margin: 0;} （不建议）淘宝的样式初始化：
body, h1, h2, h3, h4, h5, h6, hr, p, blockquote, dl, dt, dd, ul, ol, li, pre, form, fieldset, legend, button, input, textarea, th, td { margin:0; padding:0; }
body, button, input, select, textarea { font:12px/1.5tahoma, arial, \5b8b\4f53; }
h1, h2, h3, h4, h5, h6{ font-size:100%; }
address, cite, dfn, em, var { font-style:normal; }
code, kbd, pre, samp { font-family:couriernew, courier, monospace; }

31.canvas 在标签上设置宽高 和在 style 中设置宽高有什么区别：

canvas 标签的 width 和 height 是画布实际宽度和高度，绘制的图形都是在这个上面。而 style 的 width 和 height 是 canvas 在浏览器中被渲染的高度和宽度。如果 canvas 的 width 和 height 没指定或值不正确，就被设置成默认值。

32. 什么是 css HACK？

CSS hack 是通过在 CSS 样式中加入一些特殊的符号，让不同的浏览器识别不同的符号，以达到应用不同的 CSS 样式的目的复制代码

33. Less/Sass/Scss 的区别

Scss 其实是 Sass 的改进版本 Scss 是 Sass 的缩排语法，对于写惯 css 前端的 web 开发者来说很不直观，也不能将 css 代码加入到 Sass 里面，因此 Sass 语法进行了改良，Sass 3 就变成了 Scss(sassy css)。与原来的语法兼容，只是用{}取代了原来的缩进。
Less 环境较 Sass 简单 Sass 的安装需要安装 Ruby 环境，Less 基于 JavaScript，需要引入 Less.js 来处理代码输出 css 变量符不一样，Less 是@，而 Sass 是$，而且变量的作用域也不一样。
Sass 没有局部变量，满足就近原则。Less 中{}内定义的变量为局部变量。
Less 没有输出设置，Sass 提供 4 中输出选项:
输出样式的风格可以有四种选择，默认为 nested nested：嵌套缩进的 css 代码 expanded：展开的多行 css 代码 compact：简洁格式的 css 代码 compressed：压缩后的 css 代码
Sass 支持条件语句，可以使用 if{}else{},for{}循环等等。而 Less 不支持。
Less 与 Sass 处理机制不一样 Less 是通过客户端处理的，Sass 是通过服务端处理，相比较之下 Less 解析会比 Sass 慢一点

Sass 和 Less 的工具库不同 Sass 有工具库 Compass, 简单说，Sass 和 Compass 的关系有点像 Javascript 和 jQuery 的关系,Compass 是 Sass 的工具库。在 它的基础上，封装了一系列有用的模块和模板，补充强化了 Sass 的功
能。
Less 有 UI 组件库 Bootstrap, Bootstrap 是 web 前端开发中一个比较有名的前端 UI 组件库，Bootstrap 的样式文件部分源码就是采用 Less 语法编写，不过 Bootstrap4 也开始用 Sass 编写了。

34. css 与 js 动画差异：

css 性能好
css 代码逻辑相对简单 js 动画控制好
js 兼容性好
js 可实现的动画多 js 可以添加事件

35. CSS 预处理器(Sass/Less/Postcss)：

CSS 预处理器的原理: 是将类 CSS 语言通过 Webpack 编译 转成浏览器可读的真正 CSS。在这层编译之上，便可以赋予 CSS 更多更强大的功能，常用功能:
嵌套
变量
循环语句
条件语句
自动前缀
单位转换
mixin 复用

36.CSS 动画：

transition: 过渡动画
transition-property: 属性 transition-duration: 间隔 transition-timing-function: 曲线 transition-delay: 延迟
常用钩子: transitionend
2. animation / keyframes
animation-name: 动画名称，对应 @keyframes animation-duration: 间隔 animation-timing-function: 曲线 animation-delay: 延迟 animation-iteration-count: 次数
infinite: 循环动画
animation-direction: 方向

alternate: 反向播放
animation-fill-mode: 静止模式
forwards: 停止时，保留最后一帧
backwards: 停止时，回到第一帧
both: 同时运用 forwards / backwards
常用钩子: animationend
动画属性: 尽量使用动画属性进行动画，能拥有较好的性能表现：
translate scale rotate skew opacity color

37.去除浮动影响，防止父级高度塌陷：

通过增加尾元素清除浮动
:after /
: clear: both
创建父级 BFC 父级设置高度

38.选择器优先级：

!important > 行内样式 > #id > .class > tag > * > 继承 > 默认选择器 从右往左 解析

39.居中布局：

水平居中
行内元素: text-align: center 块级元素: margin: 0 auto absolute + transform
flex + justify-content: center
垂直居中
line-height: height absolute + transform flex + align-items: center table

40.层叠上下文：

元素提升为一个比较特殊的图层，在三维空间中 (z 轴) 高出普通元素一等。
触发条件
根层叠上下文(html)
position
css3 属性
flex transform opacity filter will-change

-webkit-overflow-scrolling
层叠等级：层叠上下文在 z 轴上的排序
在同一层叠上下文中，层叠等级才有意义
z-index 的优先级最高

41.BFC：

块级格式化上下文，是一个独立的渲染区域，让处于 BFC 内部的元素与外部的元素相互隔离，使内外元素的定位不会相互影响。
IE 下为 Layout，可通过 zoom:1 触发
触发条件: o 根元素
o position: absolute/fixed
o display: inline-block / table
o float 元素
o ovevflow !== visible
规则:
o 属于同一个 BFC 的两个相邻 Box 垂直排列
o 属于同一个 BFC 的两个相邻 Box 的 margin 会发生重叠
o BFC 中子元素的 margin box 的左边， 与包含块 (BFC) border box 的左边相接触 (子元素 absolute 除外)
o BFC 的区域不会与 float 的元素区域重叠

o 计算 BFC 的高度时，浮动子元素也参与计算 o 文字层不会被浮动层覆盖，环绕于周围
应用:
o 阻止 margin 重叠
o 可以包含浮动元素 —— 清除内部浮动(清除浮动的原理是两个 div 都位于同一个 BFC 区域之中) o 自适应两栏布局
o 可以阻止元素被浮动元素覆盖

42.介绍一下标准的 CSS 的盒子模型？与低版本 IE 的盒子模型有什么不同的？

标准盒子模型：宽度=内容的宽度（content）+ border + padding + margin 低版本 IE 盒子模型：宽度=内容宽度（content+border+padding）+ margin

43.box-sizing 属性？

用来控制元素的盒子模型的解析模式，默认为 content-box context-box：W3C 的标准盒子模型，设置元素的 height/width 属性指的是 content 部分的高/宽 border-box：IE 传统盒子模型。设置元素的 height/width 属性指的是 border + padding + content 部分的高/宽

44.CSS 选择器有哪些？哪些属性可以继承？

CSS 选择符：id 选择器(#myid)、类选择器(.myclassname)、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（*）、属性选择器（a[rel=”external”]）、伪类选择器
（a:hover, li:nth-child）
可继承的属性：font-size, font-family, color
不可继承的样式：border, padding, margin, width, height
优先级（就近原则）：!important > [id > class > tag] !important 比内联优先级高

45.CSS 优先级算法如何计算？

元素选择符：1 class 选择符：10 id 选择符：100 元素标签：1000
1. !important 声明的样式优先级最高，如果冲突再进行计算。
2. 如果优先级相同，则选择最后出现的样式。
3. 继承得到的样式的优先级最低。
4. CSS3 新增伪类有哪些?
p:first-of-type 选择属于其父元素的首个元素 p:last-of-type 选择属于其父元素的最后元素 p:only-of-type 选择属于其父元素唯一的元素 p:only-child 选择属于其父元素的唯一子元素 p:nth-child(2) 选择属于其父元素的第二个子元素:enabled :disabled 表单控件的禁用状态。:checked 单选框或复选框被选中。

46.如何居中 div？如何居中一个浮动元素？如何让绝对定位的 div 居中？

div：
border: 1px solid red;

margin: 0 auto;
height: 50px;
width: 80px;
浮动元素的上下左右居中：
border: 1px solid red;
float: left;
position: absolute;
width: 200px;
height: 100px;
left: 50%;
top: 50%;
margin: -50px 0 0 -100px;
绝对定位的左右居中：
border: 1px solid black;
position: absolute;
width: 200px;
height: 100px;
margin: 0 auto;
left: 0;
right: 0;

47.display 有哪些值？说明他们的作用？

inline（默认）－内联 none－隐藏 block－块显示 table－表格显示 list-item－项目列表 inline-block

48.position 的值？

static（默认）：按照正常文档流进行排列；
relative（相对定位）：不脱离文档流，参考自身静态位置通过 top, bottom, left, right 定位；
absolute(绝对定位)：参考距其最近一个不为 static 的父级元素通过 top, bottom, left, right 定位；
fixed(固定定位)：所固定的参照对像是可视窗口。

49. 文字阴影：

text-shadow：5px 5px 5px #FF0000;（水平阴影，垂直阴影，模糊距离，阴影颜色）

50.font-face 属性：

定义自己的字体

51.圆角（边框半径）：

border-radius 属性用于创建圆角

52.边框图片：

border-image: url(border.png) 30 30 round

53.盒阴影：

box-shadow: 10px 10px 5px #888888

54.媒体查询：

定义两套 css，当浏览器的尺寸变化时会采用不同的属性

55.请解释一下 CSS3 的 flexbox（弹性盒布局模型）,以及适用场景？

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。 使用场景：弹性布局适合于移动前端开发，在 Android 和 ios 上也完美支持。

56.用纯 CSS 创建一个三角形的原理是什么？

首先，需要把元素的宽度、高度设为 0。然后设置边框样式。
width: 0;
height: 0;
border-top: 40px solid transparent;
border-left: 40px solid transparent;
border-right: 40px solid transparent;
border-bottom: 40px solid #ff0000;

57.一个满屏品字布局如何设计？

第一种真正的品字：
1. 三块高宽是确定的；
2. 上面那块用 margin: 0 auto;居中；
3. 下面两块用 float 或者 inline-block 不换行；
4. 用 margin 调整位置使他们居中。
第二种全屏的品字布局：上面的 div 设置成 100%，下面的 div 分别宽 50%，然后使用 float 或者 inline 使其不换行。

58.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。

59.absolute 的 containing block 计算方式跟正常流有什么不同？

无论属于哪种，都要先找到其祖先元素中最近的 position 值不为 static 的元素，然后再判断：
1. 若此元素为 inline 元素，则 containing block 为能够包含这个元素生成的第一个和最后一个 inline box 的
padding box (除 margin, border 外的区域) 的最小矩形；
2. 否则,则由这个祖先元素的 padding box 构成。

如果都找不到，则为 initial containing block。
补充：
1. static(默认的)/relative：简单说就是它的父元素的内容框（即去掉 padding 的部分）
2. absolute: 向上找最近的定位为 absolute/relative 的元素
3. fixed: 它的 containing block 一律为根元素(html/body)

60.解释 css sprites ，如何使用？

CSS Sprites 其实就是把网页中一些背景图片整合到一张图片文件中，
再利用 CSS 的“background-image”，“background- repeat”，“background-position”的组合进行背景定位，background-position 可以用数字能精确的定位出背景图片的位置。
CSS Sprites 为一些大型的网站节约了带宽，让提高了用户的加载速度和用户体验，不需要加载更多的图片

61.阐述一下 CSS Sprites：

将一个页面涉及到的所有图片都包含到一张大图中去，然后利用 CSS 的 background-image，background- repeat，background-position 的组合进行背景定位。利用 CSS Sprites 能很好地减少网页的 http 请求，从而大大的提高页面的性能；CSS Sprites 能减少图片的字节。
62.CSS 属性 overflow 属性定义溢出元素内容区的内容会如何处理?

参数是 scroll 时候，必会出现滚动条。
参数是 auto 时候，子元素内容大于父元素时出现滚动条。参数是 visible 时候，溢出的内容出现在父元素之外。
参数是 hidden 时候，溢出隐藏。

63.style 标签写在 body 后与 body 前有什么区别

页面加载自上而下 当然是先加载样式。 写在 body 标签后由于浏览器以逐行方式对 HTML 文档进行解析，当解析到写在尾部的样式表（外联或写在 style 标签）会导致浏览器停止之前的渲染，等待加载且解析样式表完成之后重新渲染，在 windows 的 IE 下可能会出现 FOUC 现象（即样式失效导致的页面闪烁问题）

64.png、jpg、gif 这些图片格式解释一下，分别什么时候用。有没有了解过 webp

png 是便携式网络图片（Portable Network Graphics）是一种无损数据压缩位图文件格式.优点是：压缩比高，色彩好。 大多数地方都可以用。
jpg 是一种针对相片使用的一种失真压缩方法，是一种破坏性的压缩，在色调及颜色平滑变化做的不错。在 www 上，被用来储存和传输照片的格式。
gif 是一种位图文件格式，以 8 位色重现真色彩的图像。可以实现动画效果.
webp 格式是谷歌在 2010 年推出的图片格式，压缩率只有 jpg 的 2/3，大小比 png 小了 45%。缺点是压缩的时间更久了，兼容性不好，目前谷歌和 opera 支持
有一个高度自适应的 div，里面有两个 div，一个高度 100px，希望另一个填满剩下的高度
外层 div 使用 position：relative；高度要求自适应的 div 使用 position: absolute; top: 100px; bottom: 0; left: 0

65.display:inline-block 什么时候会显示间隙？

有空格时候会有间隙 解决：移除空格
margin 正值的时候 解决：margin 使用负值
使用 font-size 时候 解决：font-size:0、letter-spacing、word-spacing

66. li 与 li 之间有看不见的空白间隔是什么原因引起的？有什么解决办法？

可以将
代码全部写在一排
浮动 li 中 float：left
在 ul 中用 font-size：0（谷歌不支持）；可以使用 letter-space：-3px

67.如果需要手动写动画，你认为最小时间间隔是多久，为什么？

多数显示器默认频率是 60Hz，即 1 秒刷新 60 次，所以理论上最小间隔为 1/60＊1000ms ＝ 16.7ms。

68.让页面里的字体变清晰，变细用 CSS 怎么做？

-webkit-font-smoothing 在 window 系统下没有起作用，但是在 IOS 设备上起作用 -webkit-font-smoothing：antialiased 是最佳的，灰度平滑。

69.你对 line-height 是如何理解的？

行高是指一行文字的高度，具体说是两行文字间基线的距离。CSS 中起高度作用的是 height 和 line-height，没有定义 height 属性，最终其表现作用一定是 line-height。
单行文本垂直居中：把 line-height 值设置为 height 一样大小的值可以实现单行文字的垂直居中，其实也可以把 height 删除。 多行文本垂直居中：需要设置 display 属性为 inline-block。

70.::before 和 :after 中双冒号和单冒号有什么区别？解释一下这 2 个伪元素的作用

单冒号(:)用于 CSS3 伪类，双冒号(::)用于 CSS3 伪元素。
::before 就是以一个子元素的存在，定义在元素主体内容之前的一个伪元素。并不存在于 dom 之中，只存在在页面之中。
:before 和 :after 这两个伪元素，是在 CSS2.1 里新出现的。起初，伪元素的前缀使用的是单冒号语法，但随着 Web 的进化，在 CSS3 的规范里，伪元素的语法被修改成使用双冒号，成为::before ::after

71.视差滚动效果？

视差滚动（Parallax Scrolling）通过在网页向下滚动的时候，控制背景的移动速度比前景的移动速度慢来创建
出令人惊叹的 3D 效果。
CSS3 实现 优点：开发时间短、性能和开发效率比较好，缺点是不能兼容到低版本的浏览器
jQuery 实现 通过控制不同层滚动速度，计算每一层的时间，控制滚动效果。 优点：能兼容到各个版本的，效果可控性好 缺点：开发起来对制作者要求高

插件实现方式 例如：parallax-scrolling，兼容性十分好

72.什么是响应式设计？响应式设计的基本原理是什么？如何兼容低版本的 IE？

响应式网站设计(Responsive Web design)是一个网站能够兼容多个终端，而不是为每一个终端做一个特定的版本。 基本原理是通过媒体查询检测不同的设备屏幕尺寸做处理。 页面头部必须有 meta 声明的 viewport。

73.全屏滚动的原理是什么？用到了 CSS 的哪些属性？

原理：有点类似于轮播，整体的元素一直排列下去，假设有 5 个需要展示的全屏页面，那么高度是 500%，只是展示 100%，剩下的可以通过 transform 进行 y 轴定位，也可以通过 margin-top 实现
overflow：hidden；transition：all 1000ms ease；

74.元素竖向的百分比设定是相对于容器的高度吗？

当按百分比设定一个元素的宽度时，它是相对于父容器的宽度计算的，但是，对于一些表示竖向距离的属性，例如 padding-top , padding-bottom , margin-top , margin-bottom 等，当按百分比设定它们时，依据的也是父容器的宽度，而不是高度。

75.margin 和 padding 分别适合什么场景使用？

何时使用 margin：
需要在 border 外侧添加空白空白处不需要背景色
上下相连的两个盒子之间的空白，需要相互抵消时。
何时使用 padding：
需要在 border 内侧添加空白空白处需要背景颜色
上下相连的两个盒子的空白，希望为两者之和。
兼容性的问题：在 IE5 IE6 中，为 float 的盒子指定 margin 时，左侧的 margin 可能会变成两倍的宽度。通过改变 padding 或者指定盒子的 display：inline 解决。

76.在网页中的应该使用奇数还是偶数的字体？为什么呢？

使用偶数字体。偶数字号相对更容易和 web 设计的其他部分构成比例关系。Windows 自带的点阵宋体（中易宋体）从 Vista 开始只提供 12、14、16 px 这三个大小的点阵，而 13、15、17 px 时用的是小一号的点。（即每个字占的空间大了 1 px，但点阵没变），于是略显稀疏。

77.浏览器是怎样解析 CSS 选择器的？

CSS 选择器的解析是从右向左解析的。若从左向右的匹配，发现不符合规则，需要进行回溯，会损失很多性能。
若从右向左匹配，先找到所有的最右节点，对于每一个节点，向上寻找其父节点直到找到根元素或满足条件的
匹配规则，则结束这个分支的遍历。
两种匹配规则的性能差别很大，是因为从右向左的匹配在第一步就筛选掉了大量的不符合条件的最右节点（叶
子节点），而从左向右的匹配规则的性能都浪费在了失败的查找上面。
而在 CSS 解析完毕后，需要将解析的结果与 DOM Tree 的内容一起进行分析建立一棵 Render Tree，最终用来进行绘图。
在建立 Render Tree 时（WebKit 中的「Attachment」过程），浏览器就要为每个 DOM Tree 中的元素根据 CSS 的解析结果（Style Rules）来确定生成怎样的 Render Tree。

78.CSS 优化、提高性能的方法有哪些？

避免过度约束
避免后代选择符避免链式选择符使用紧凑的语法
避免不必要的命名空间
避免不必要的重复
最好使用表示语义的名字。一个好的类名应该是描述他是什么而不是像什么
避免！important，可以选择其他选择器
尽可能的精简规则，你可以合并不同类里的重复规则

79.使用 CSS 预处理器吗？

Less sass

80.移动端的布局用过媒体查询吗？

通过媒体查询可以为不同大小和尺寸的媒体定义不同的 css，适应相应的设备的显示里边
CSS : @media only screen and (max-device-width:480px) {/css 样式/}、

81.设置元素浮动后，该元素的 display 值是多少？

自动变成 display:block

82.上下 margin 重合的问题

在重合元素外包裹一层容器，并触发该容器生成一个 BFC。 例子：

```
<div class="aside"></div> <div class="text">
<div class="main"></div>
</div>
<!--下面是 css 代码-->
.aside {
margin-bottom: 100px;
width: 100px; height: 150px; background: #f66;
}
.main {
margin-top: 100px; height: 200px; background: #fcc;
}
.text{
/*盒子 main 的外面包一个 div，通过改变此 div 的属性使两个
盒子分属于两个不同的 BFC，
以此来阻止 margin 重叠*/overflow: hidden; //此时已经触发了 BFC 属性。
}
```

83.为什么会出现浮动和什么时候需要清除浮动？清除浮动的方式？

浮动元素碰到包含它的边框或者浮动元素的边框停留。由于浮动元素不在文档流中，所以文档流的块框表现得
就像浮动框不存在一样。
浮动元素会漂浮在文档流的块框上。 浮动带来的问题：
父元素的高度无法被撑开，影响与父元素同级的元素
与浮动元素同级的非浮动元素（内联元素）会跟随其后
若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构。
清除浮动的方式：
父级 div 定义 height
最后一个浮动元素后加空 div 标签 并添加样式 clear:both。
包含浮动元素的父标签添加样式 overflow 为 hidden 或 auto。父级 div 定义 zoo

84.对 BFC 规范(块级格式化上下文：block formatting context)的理解？

BFC 规定了内部的 Block Box 如何布局。 定位方案：
内部的 Box 会在垂直方向上一个接一个放置。
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。BFC 的区域不会与 float box 重叠。
BFC 是页面上的一个隔离的独立容器，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，浮动元素也会参与计算。
满足下列条件之一就可触发 BFC
根元素，即 html
float 的值不为 none（默认）overflow 的值不为 visible（默认）
display 的值为 inline-block、table-cell、table-caption position 的值为 absolute 或 fixed

85.position 跟 display、overflow、float 这些特性相互叠加后会怎么样？

display 属性规定元素应该生成的框的类型；position 属性规定元素的定位类型；float 属性是一种布局方式，定义元素在哪个方向浮动。
类似于优先级机制：position：absolute/fixed 优先级最高，有他们在时，float 不起作用，display 值需要调整。float 或者 absolute 定位的元素，只能是块元素或表格。

86.CSS 里的 visibility 属性有个 collapse 属性值？在不同浏览器下以后什么区别？

当一个元素的 visibility 属性被设置成 collapse 值后，对于一般的元素，它的表现跟 hidden 是一样的。
chrome 中，使用 collapse 值和使用 hidden 没有区别。
firefox，opera 和 IE，使用 collapse 值和使用 display：none 没有什么区别。

87.absolute 的 containing block 计算方式跟正常流有什么不同？

无论属于哪种，都要先找到其祖先元素中最近的 position 值不为 static 的元素，然后再判断：
若此元素为 inline 元素，则 containing block 为能够包含这个元素生成的第一个和最后一个 inline box 的 padding box (除 margin, border 外的区域) 的最小矩形；
否则,则由这个祖先元素的 padding box 构成。
如果都找不到，则为 initial containing block。
补充：
static(默认的)/relative：简单说就是它的父元素的内容框（即去掉 padding 的部分）absolute: 向上找最近的定位为 absolute/relative 的元素

88.常见的兼容性问题？

不同浏览器的标签默认的 margin 和 padding 不一样。
*{margin:0;padding:0;}
IE6 双边距 bug：块属性标签 float 后，又有横行的 margin 情况下，在 IE6 显示 margin 比设置的大。hack：display:inline;将其转化为行内属性。
渐进识别的方式，从总体中逐渐排除局部。首先，巧妙的使用“9”这一标记，将 IE 浏览器从所有情况中分离出来。接着，再次使用“+”将 IE8 和 IE7、IE6 分离开来，这样 IE8 已经独立识别。
{background-color:#f1ee18;
/*所有识别*/
.background-color:#00deff\9; /*IE6、7、8 识别*/ +background-color:#a200ff; /*IE6、7 识别*/ _background-color:#1e0bd1; /*IE6 识别*/}
12345678

设置较小高度标签（一般小于 10px），在 IE6，IE7 中高度超出自己设置高度。hack：给超出高度的标签设置 overflow:hidden;或者设置行高 line-height 小于你设置的高度。
IE 下，可以使用获取常规属性的方法来获取自定义属性,也可以使用 getAttribute()获取自定义属性；Firefox 下，
只能使用 getAttribute()获取自定义属性。解决方法:统一通过 getAttribute()获取自定义属性。
Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示,可通过加入 CSS 属性 -webkit-text-size- adjust: none; 解决。
超链接访问过后 hover 样式就不出现了，被点击访问过的超链接样式不再具有 hover 和 active 了。
解决方法是改变 CSS 属性的排列顺序:L-V-H-A (love hate): a:link {} a:visited {} a:hover {} a:active {}

89.请解释一下 CSS3 的 flexbox（弹性盒布局模型）,以及适用场景？

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。

90.CSS3 有哪些新特性？

RGBA 和透明度
background-image background-origin(content-box/padding-box/border-box) background-size background-repeat
word-wrap（对长的不可分割单词换行）word-wrap：break-word
文字阴影：text-shadow： 5px 5px 5px #FF0000;（水平阴影，垂直阴影，模糊距离，阴影颜色）font-face 属性：定义自己的字体
圆角（边框半径）：border-radius 属性用于创建圆角
边框图片： border-image: url(border.png) 30 30 round 盒阴影： box-shadow: 10px 10px 5px #888888
媒体查询：定义两套 css，当浏览器的尺寸变化时会采用不同的属性
small{ font-size:12px; }
ul, ol { list-style:none; }
a { text-decoration:none; }
a:hover { text-decoration:underline; } sup { vertical-align:text-top; }
sub{ vertical-align:text-bottom; } legend { color:#000; }
fieldset, img { border:0; }

button, input, select, textarea { font-size:100%; }
table { border-collapse:collapse; border-spacing:0; }复制代码

90.解释下 CSS sprites,以及你要如何在页面或网站中使用它：

CSS Sprites 其实就是把网页中一些背景图片整合到一张图片文件中，再利用 CSS 的“background-
image”，“background- repeat”，“background-position”的组合进行背景定位，background-position 可以用数字能精确的定位出背景图片的位置。这样可以减少很多图片请求的开销，因为请求耗时比较长；请求虽然可以并发，但是也有限制，一般浏览器都是 6 个。对于未来而言，就不需要这样做了，因为有了 http2 。

91.一个页面从输入 URL 到页面加载显示完成，这个过程都发生了什么：

分为 4 个步骤：
（1），当发送一个 URL 请求时，不管这个 URL 是 Web 页面的 URL 还是 Web 页面上每个资源的 URL，浏览器都会开启一个线程来处理这个请求，同时在远程 DNS 服务器上启动一个 DNS 查询。这能使浏览器获得请求对应的 IP 地址。
（2），浏览器与远程 Web 服务器通过 TCP 三次握手协商来建立一个 TCP/IP 连接。该握手包括一个同步报文，一个同步-应答报文和一个应答报文，这三个报文在 浏览器和服务器之间传递。该握手首先由客户端尝试建立起通信，而后服务器应答并接受客户端的请求，最后由客户端发出该请求已经被接受的报文。
（3），一旦 TCP/IP 连接建立，浏览器会通过该连接向远程服务器发送 HTTP 的 GET 请求。远程服务器找到资源并使用 HTTP 响应返回该资源，值为 200 的 HTTP 响应状态表示一个正确的响应。
（4），此时，Web 服务器提供资源服务，客户端开始下载资源。请求返回后，便进入了我们关注的前端模块
简单来说，浏览器会解析 HTML 生成 DOM Tree，其次会根据 CSS 生成 CSS Rule Tree，而 javascript 又可以根据 DOM API 操作 DOM 复制代码

92.哪些地方会出现 CSS 堵塞，哪些地方会出现 JS 堵塞：

js 的阻塞特性：所有浏览器在下载 JS 下载、解析、执行完毕后才开始继续 JS，但是 由于浏览器为了防止出现 DOM 树，需要重新构建 嵌入 JS 只会阻塞其后内容的显示，2 种方式都会阻塞其后资源的下载。也就是说外部样式不会阻塞外部脚本的加载，但会阻塞外部脚本的执行。
CSS 本来是可以并行下载的，在什么情况下会出现阻塞加载了(在测试观察中，CSS 都是阻塞加载）
当 JS 的时候，该 JS 放到 根本原因：因为浏览器会维持 css 和 JS 会阻塞后面的资源加载，所以就会出现上面 嵌入
1、放在底部，虽然放在底部照样会阻塞所有呈现，但不会阻塞资源下载。 2、如果嵌入 JS 放在 head 中，请把嵌入 JS 放在 CSS 头部。 3、使用 defer（只支持 IE） 4、不要在嵌入的 JS 中调用运行时间较长的函数，如果一定要
用，可以用 setTimeout 来调用

前端面试题集锦——JavaScript

1.请你谈谈 Cookie 的优缺点

优点：极高的扩展性和可用性
1）数据持久性。
2）不需要任何服务器资源。 Cookie 存储在客户端并在发送后由服务器读取。
3）可配置到期规则。 控制 cookie 的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的 cookie 。
4）简单性。 基于文本的轻量结构。
5）通过良好的编程，控制保存在 cookie 中的 session 对象的大小。
6）通过加密和安全传输技术（ SSL ），减少 cookie 被破解的可能性。
7）只在 cookie 中存放不敏感数据，即使被盗也不会有重大损失。
缺点：
1）Cookie 数量和长度的限制 。
数量：每个域的 cookie 总数有限。
a）IE6 或更低版本最多 20 个 cookie
b）IE7 和之后的版本最后可以有 50 个 cookie
c）Firefox 最多 50 个 cookie
d）chrome 和 Safari 没有做硬性限制
长度：每个 cookie 长度不超过 4KB （ 4096B ），否则会被截掉。
2）潜在的安全风险 。 Cookie 可能被拦截、篡改。如果 cookie 被拦截，就有可能取得所有的 session 信息。
3）用户配置为禁用 。有些用户禁用了浏览器或客户端设备接受 cookie 的能力，因此限制了这一功能。

4) 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，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

2.Array.prototype.slice.call(arr，2)方法的作用是：

利用 Array 原型上的 slice 方法，使用 call 函数的第一个参数，让这个方法中的 this 指向 arr，并传递参数 2，实际上等于 arr.slice(2)，即从下标为 2 开始截取到末尾。

3.以下代码执行后，控制台的输出是：

```
var  a  =  10;
function  a(){}
console.log(typeof  a)
A.&amp;quot;number&amp;quot;
B.&amp;quot;object&amp;quot;
C.&amp;quot;function&amp;quot;
D.&amp;quot;undefined&amp;quot
```

答案：C
函数提升优先级高于变量提升，所以代码等价于

```
function a(){}
var a;
a = 10;
console.log(typeof a)
```

4、简单说一下浏览器本地存储是怎样的

总的来说，浏览器存储分为以下几种：
1、Cookie 存储，明文，大小限制 4k 等
2、localStorage，持久化存储方式之一，不用在两端之间传输，且限制大小为 10M
3、sessionStorage，会话级存储方式，浏览器关闭立即数据丢失
4、indexDb，浏览器端的数据库

5.原型 / 构造函数 / 实例

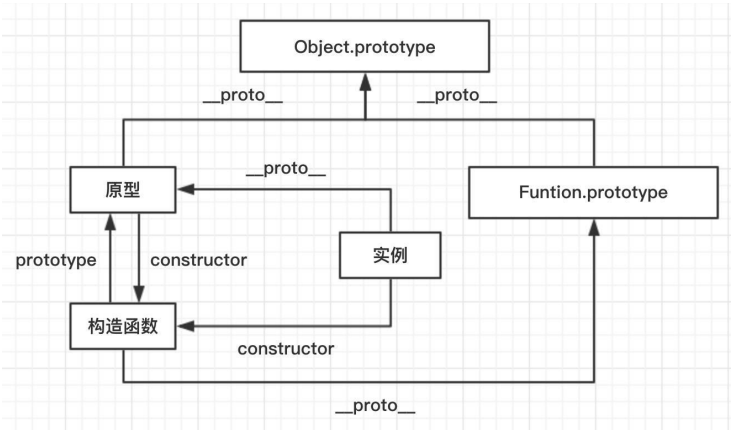
原型(prototype): 一个简单的对象，用于实现对象的 属性继承。可以简单的理解成对象的爹。在 Firefox 和 Chrome 中，每个 JavaScript 对象中都包含一个__proto__ (非标准)的属性指向它爹(该对象的原型)，可 obj.__proto__进行访问。

构造函数: 可以通过 new 来 新建一个对象 的函数。

实例: 通过构造函数和 new 创建出来的对象，便是实例。 实例通过__proto__指向原型，通过 constructor 指向构造函数。

说了一大堆，大家可能有点懵逼，这里来举个栗子，以 Object 为例，我们常用的 Object 便是一个构造函数，因此我们可以通过它构建实例。

// 实例 const instance = new Object()复制代码
则此时， 实例为 instance, 构造函数为 Object，我们知道，构造函数拥有一个 prototype 的属性指向原型，因此原型为:
// 原型 const prototype = Object.prototype
这里我们可以来看出三者的关系:
实例.__proto__ === 原型
原型.constructor === 构造函数
构造函数.prototype === 原型
// 这条线其实是是基于原型进行获取的，可以理解成一条基于原型的映射线// 例如: // const o = new Object()// o.constructor === Object --> true// o.__proto__ = null;// o.constructor === Object --> false// 注意: 其实实例上并不是真正有 constructor 这个指针，它其实是从原型链上获取的// instance.hasOwnProperty('constructor') === false (感谢 刘博海 Brian 童鞋
实例.constructor === 构造函数
此处感谢 caihaihong 童鞋的指出。
放大来看，我画了张图供大家彻底理解:



6.原型链：

原型链是由原型对象组成，每个对象都有 __proto__ 属性，指向了创建该对象的构造函数的原型，__proto__ 将对象连接起来组成了原型链。是一个用来实现继承和共享属性的有限的对象链。

属性查找机制: 当查找对象的属性时，如果实例对象自身不存在该属性，则沿着原型链往上一级查找，找到时则输出，不存在时，则继续沿着原型链往上一级查找，直至最顶级的原型对象 Object.prototype，如还是没找到，则输出 undefined;

属性修改机制: 只会修改实例对象本身的属性,如果不存在,则进行添加该属性，如果需要修改原型的属性时，则可以用: b.prototype.x = 2；但是这样会造成所有继承于该对象的实例的属性发生改变。

7.执行上下文(EC)

执行上下文可以简单理解为一个对象:
它包含三个部分:
o 变量对象(VO)
o 作用域链(词法作用域)
othis 指向
它的类型:
o 全局执行上下文
o 函数执行上下文
oeval 执行上下文
代码执行过程:
o 创建 全局上下文 (global EC)
o 全局执行上下文 (caller) 逐行 自上而下 执行。遇到函数时，函数执行上下文 (callee) 被 push 到执行栈顶层
o 函数执行上下文被激活，成为 active EC, 开始执行函数中的代码，caller 被挂起
o 函数执行完后，callee 被 pop 移除出执行栈，控制权交还全局上下文 (caller)，继续执行。

8.变量对象

变量对象，是执行上下文中的一部分，可以抽象为一种 数据作用域，其实也可以理解为就是一个简单的对象，它存储着该执行上下文中的所有 变量和函数声明(不包含函数表达式)。
活动对象 (AO): 当变量对象所处的上下文为 active EC 时，称为活动对象。

9.作用域链

我们知道，我们可以在执行上下文中访问到父级甚至全局的变量，这便是作用域链的功劳。作用域链可以理解为一组对象列表，包含 父级和自身的变量对象，因此我们便能通过作用域链访问到父级里声明的变量或者函数。
由两部分组成:
o[[scope]]属性: 指向父级变量对象和作用域链，也就是包含了父级的[[scope]]和 AO
oAO: 自身活动对象
如此 [[scopr]]包含[[scope]]，便自上而下形成一条 链式作用域。

10.闭包

闭包属于一种特殊的作用域，称为 静态作用域。它的定义可以理解为: 父函数被销毁 的情况下，返回出的子函数的[[scope]]中仍然保留着父级的单变量对象和作用域链，因此可以继续访问到父级的变量对象，这样的函数称为闭包。
闭包会产生一个很经典的问题:
o 多个子函数的[[scope]]都是同时指向父级，是完全共享的。因此当父级的变量对象被修改时，所有子函数都受到影响。
解决:
o 变量可以通过 函数参数的形式 传入，避免使用默认的[[scope]]向上查找
o 使用 setTimeout 包裹，通过第三个参数传入
o 使用 块级作用域，让变量成为自己上下文的属性，避免共享
6. script 引入方式：
html 静态<script>引入
js 动态插入<script>
<script defer>: 延迟加载，元素解析完成后执行
<script async>: 异步加载，但执行时会阻塞元素渲染

11.对象的拷贝

浅拷贝：以赋值的形式拷贝引用对象，仍指向同一个地址，修改时原对象也会受到影响
oObject.assign
o 展开运算符(...)
深拷贝：完全拷贝一个新对象，修改时原对象不再受到任何影响
oJSON.parse(JSON.stringify(obj))：性能最快
  具有循环引用的对象时，报错
  当值为函数、undefined、或 symbol 时，无法拷贝
o 递归进行逐一赋值

12.new 运算符的执行过程

新生成一个对象
链接到原型：obj.__proto__ = Con.prototype
绑定 this：apply
返回新对象(如果构造函数有自己 retrun 时，则返回该值)

13.instanceof 原理

能在实例的 原型对象链 中找到该构造函数的 prototype 属性所指向的 原型对象，就返回 true。即：

```
// __proto__：代表原型对象链
instance.[__proto__...] === instance.constructor.prototype
// return true 复制代码
```

14.代码的复用

当你发现任何代码开始写第二遍时，就要开始考虑如何复用。一般有以下的方式：
函数封装
继承
复制 extend
混入 mixin
借用 apply/call

15.继承

在 JS 中，继承通常指的便是 原型链继承，也就是通过指定原型，并可以通过原型链继承原型上的属性或者方法。
最优化: 圣杯模式

```
var inherit = (function(c,p){
    var F = function(){};
    return function(c,p){
        F.prototype = p.prototype;
        c.prototype = new F();
        c.uber = p.prototype;
        c.prototype.constructor = c;
    }
})();
```

使用 ES6 的语法糖 class / extends

16.类型转换

大家都知道 JS 中在使用运算符号或者对比符时，会自带隐式转换，规则如下:
  -、*、/、% ：一律转换成数值后计算
  +:
o 数字 + 字符串 = 字符串， 运算顺序是从左到右
o 数字 + 对象， 优先调用对象的 valueOf -> toString
o 数字 + boolean/null -> 数字
o 数字 + undefined -> NaN
  [1].toString() === '1'
  {}.toString() === '[object object]'
  NaN !== NaN 、+undefined 为 NaN

17.类型判断

判断 Target 的类型，单单用 typeof 并无法完全满足，这其实并不是 bug，本质原因是 JS 的万物皆对象的理论。因此要真正完美判断时，我们需要区分对待:
基本类型(null): 使用 String(null)
基本类型(string / number / boolean / undefined) + function: 直接使用 typeof 即可
其余引用类型(Array / Date / RegExp Error): 调用 toString 后根据[object

XXX]进行判断
很稳的判断封装:

```
let class2type = {}'Array Date RegExp Object Error'.split(' ').forEach(e => class2type[ '[object ' + e + ']' ] = e.toLowerCase())
function type(obj) {
    if (obj == null) return String(obj)
    return typeof obj === 'object' ? class2type[ Object.prototype.toString.call(obj) ] || 'object' : typeof obj
}
```

18.模块化

模块化开发在现代开发中已是必不可少的一部分，它大大提高了项目的可维护、可拓展和可协作性。通常,我们 在浏览器中使用 ES6 的模块化支持,在 Node 中使用 commonjs 的模块化支持。
分类:
oes6: import / export
ocommonjs: require / module.exports / exports
oamd: require / defined
require 与 import 的区别
orequire 支持 动态导入，import 不支持，正在提案 (babel 下可支持)
orequire 是 同步 导入，import 属于 异步 导入
orequire 是 值拷贝，导出值变化不会影响导入值；import 指向 内存地址，导入值会随导出值而变化

19.防抖与节流

防抖与节流函数是一种最常用的 高频触发优化方式，能对性能有较大的帮助。
防抖 (debounce): 将多次高频操作优化为只在最后一次执行，通常使用的场景是：用户输入，只需再输入完成后做一次输入校验即可。

```
function debounce(fn, wait, immediate) {
    let timer = null

    return function() {
        let args = arguments
        let context = this

        if (immediate && !timer) {
```

```
            fn.apply(context, args)
        }

        if (timer) clearTimeout(timer)
        timer = setTimeout(() => {
            fn.apply(context, args)
        }, wait)
    }
}
```

节流(throttle): 每隔一段时间后执行一次，也就是降低频率，将高频操作优化成低频操作,通常使用场景: 滚动条事件 或者 resize 事件,通常每隔 100~500 ms 执行一次即可。

```
function throttle(fn, wait, immediate) {
    let timer = null
    let callNow = immediate

    return function() {
        let context = this,
            args = arguments

        if (callNow) {
            fn.apply(context, args)
            callNow = false
        }

        if (!timer) {
            timer = setTimeout(() => {
                fn.apply(context, args)
                timer = null
            }, wait)
        }
    }
}
```

20.函数执行改变 this

由于 JS 的设计原理: 在函数中，可以引用运行环境中的变量。因此就需要一个机制来让我们可以在函数体内部获取当前的运行环境，这便是 this。
因此要明白 this 指向，其实就是要搞清楚 函数的运行环境，说人话就是，谁调用了函数。例如:

obj.fn()，便是 obj 调用了函数，既函数中的 this === obj
fn()，这里可以看成 window.fn()，因此 this === window

但这种机制并不完全能满足我们的业务需求，因此提供了三种方式可以手动修改 this 的指向:

- call: fn.call(target, 1, 2)
- apply: fn.apply(target, [1, 2])
- bind: fn.bind(target)(1,2)

21.ES6/ES7

由于 Babel 的强大和普及，现在 ES6/ES7 基本上已经是现代化开发的必备了。通过新的语法糖，能让代码整体更为简洁和易读。

声明

olet / const: 块级作用域、不存在变量提升、暂时性死区、不允许重复声明

oconst: 声明常量，无法修改

解构赋值

class / extend: 类声明与继承

Set / Map: 新的数据结构

异步解决方案:

Promise 的使用与实现

generator:

- yield: 暂停代码
- next(): 继续执行代码

```
function* helloWorld() {
  yield 'hello';
  yield 'world';
  return 'ending';
}
const generator = helloWorld();

generator.next()  // { value: 'hello', done: false }

generator.next()  // { value: 'world', done: false }

generator.next()  // { value: 'ending', done: true }

generator.next()  // { value: undefined, done: true }
```

oawait / async: 是 generator 的语法糖， babel 中是基于 promise 实现。

```
async function getUserByAsync(){
   let user = await fetchUser();
   return user;
}
const user = await getUserByAsync()console.log(user)
```

22.AST

抽象语法树 (Abstract Syntax Tree)，是将代码逐字母解析成 树状对象 的形式。这是语言之间的转换、代码语法检查，代码风格检查，代码格式化，代码高亮，代码错误提示，代码自动补全等等的基础。例如:

```
function square(n){
    return n * n
}
```

通过解析转化成的 AST 如下图:

23.babel 编译原理

babylon 将 ES6/ES7 代码解析成 AST
babel-traverse 对 AST 进行遍历转译，得到新的 AST
新 AST 通过 babel-generator 转换成 ES5

24.函数柯里化

在一个函数中，首先填充几个参数，然后再返回一个新的函数的技术，称为函数的柯里化。通常可用于在不侵入函数的前提下，为函数 预置通用参数，供多次重复调用。

```
const add = function add(x) {
    return function (y) {
        return x + y
    }
}
const add1 = add(1)
add1(2) === 3
add1(20) === 21
```

25.get 请求传参长度的误区

误区：我们经常说 get 请求参数的大小存在限制，而 post 请求的参数大小是无限制的。

实际上 HTTP 协议从未规定 GET/POST 的请求长度限制是多少。对 get 请求参数的限制是来源与浏览器或 web 服务器，浏览器或 web 服务器限制了 url 的长度。为了明确这个概念，我们必须再次强调下面几点:

HTTP 协议 未规定 GET 和 POST 的长度限制

GET 的最大长度显示是因为 浏览器和 web 服务器限制了 URI 的长度

不同的浏览器和 WEB 服务器，限制的最大长度不一样

要支持 IE，则最大长度为 2083byte，若只支持 Chrome，则最大长度 8182byte

26.补充 get 和 post 请求在缓存方面的区别

post/get 的请求区别，具体不再赘述。
补充补充一个 get 和 post 在缓存方面的区别：

get 请求类似于查找的过程，用户获取数据，可以不用每次都与数据库连接，所以可以使用缓存。

post 不同，post 做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存。因此 get 请求适合于请求缓存。

27.说一下闭包

一句话可以概括：闭包就是能够读取其他函数内部变量的函数，或者子函数在外调用，子函数所在的父函数的作用域不会被释放。

28.说一下类的创建和继承

（1）类的创建（es5）：new 一个 function，在这个 function 的 prototype 里面增加属性和方法。

下面来创建一个 Animal 类：

```
// 定义一个动物类

function Animal (name) {
```

```
// 属性
this.name = name || 'Animal';
// 实例方法
this.sleep = function(){
console.log(this.name + '正在睡觉!');
}
}
// 原型方法
Animal.prototype.eat = function(food) {
console.log(this.name + '正在吃：' + food);
};
```

这样就生成了一个 Animal 类，实力化生成对象后，有方法和属性。

（2）类的继承——原型链继承

原型链继承

```
--function Cat(){ }
Cat.prototype = new Animal();
Cat.prototype.name = 'cat';
//  Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.eat('fish'));
console.log(cat.sleep());
console.log(cat instanceof Animal); //true
console.log(cat instanceof Cat); //true
```

介绍：在这里我们可以看到 new 了一个空对象,这个空对象指向 Animal 并且 Cat.prototype 指向了这个空对象，这种就是基于原型链的继承。

特点：基于原型链，既是父类的实例，也是子类的实例

缺点：无法实现多继承

（3）构造继承：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

```
function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

// Test Code

var cat = new Cat();

console.log(cat.name);

console.log(cat.sleep());

console.log(cat instanceof Animal); // false

console.log(cat instanceof Cat); // true
```

特点：可以实现多继承

缺点：只能继承父类实例的属性和方法，不能继承原型上的属性和方法。

（4）实例继承和拷贝继承

实例继承：为父类实例添加新特性，作为子类实例返回

拷贝继承：拷贝父类元素上的属性和方法

上述两个实用性不强，不一一举例。

（5）组合继承：相当于构造继承和原型链继承的组合体。通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

```
function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

Cat.prototype = new Animal();

Cat.prototype.constructor = Cat;

// Test Code

var cat = new Cat();
```

```
console.log(cat.name);

console.log(cat.sleep());

console.log(cat instanceof Animal); // true

console.log(cat instanceof Cat); // true
```

特点：可以继承实例属性/方法，也可以继承原型属性/方法

缺点：调用了两次父类构造函数，生成了两份实例

（6）寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性

```
function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

(function(){

// 创建一个没有实例方法的类

var Super = function(){};

Super.prototype = Animal.prototype;

//将实例作为子类的原型

Cat.prototype = new Super();

})();

// Test Code

var cat = new Cat();

console.log(cat.name);

console.log(cat.sleep());

console.log(cat instanceof Animal); // true

console.log(cat instanceof Cat); //true
```

较为推荐

29.如何解决异步回调地狱

promise、generator、async/await

30.说说前端中的事件流

HTML 中与 javascript 交互是通过事件驱动来实现的，例如鼠标点击事件 onclick、页面的滚动事件 onscroll 等等，可以向文档或者文档中的元素添加事件侦听器来预订事件。想要知道这些事件是在什么时候进行调用的，就需要了解一下“事件流”的概念。
什么是事件流：事件流描述的是从页面中接收事件的顺序,DOM2 级事件流包括下面几个阶段。
事件捕获阶段
处于目标阶段
事件冒泡阶段
addEventListener：addEventListener 是 DOM2 级事件新增的指定事件处理程序的操作，这个方法接收 3 个参数：要处理的事件名、作为事件处理程序的函数和一个布尔值。最后这个布尔值参数如果是 true，表示在捕获阶段调用事件处理程序；如果是 false，表示在冒泡阶段调用事件处理程序。
IE 只支持事件冒泡。

31.如何让事件先冒泡后捕获

在 DOM 标准事件模型中,是先捕获后冒泡。但是如果要实现先冒泡后捕获的效果，对于同一个事件，监听捕获和冒泡，分别对应相应的处理函数，监听到捕获事件，先暂缓执行，直到冒泡事件被捕获后再执行捕获之间。

32.说一下事件委托

简介：事件委托指的是，不在事件的发生地（直接 dom）上设置监听函数，而是在其父元素上设置监听函数，通过事件冒泡，父元素可以监听到子元素上事件的触发，通过判断事件发生元素 DOM 的类型，来做出不同的响应。
举例：最经典的就是 ul 和 li 标签的事件监听，比如我们在添加事件时候，采用事件委托机制，不会在 li 标签上直接添加，而是在 ul 父元素上添加。

好处：比较合适动态元素的绑定，新添加的子元素也会有监听函数，也可以有事件触发机制

33.说一下图片的懒加载和预加载

预加载：提前加载图片，当用户需要查看时可直接从本地缓存中渲染。
懒加载:懒加载的主要目的是作为服务器前端的优化,减少请求数或延迟请求数。
两种技术的本质：两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。
懒加载对服务器前端有一定的缓解压力作用，预加载则会增加服务器前端压力。

34.mouseover 和 mouseenter 的区别

mouseover：当鼠标移入元素或其子元素都会触发事件，所以有一个重复触发，冒泡的过程。对应的移除事件是 mouseout
mouseenter：当鼠标移除元素本身（不包含元素的子元素）会触发事件，也就是不会冒泡，对应的移除事件是 mouseleave

35. js 的 new 操作符做了哪些事情

new 操作符新建了一个空对象，这个对象原型指向构造函数的 prototype，执行构造函数后返回这个对象。

36.改变函数内部 this 指针的指向函数（bind，apply，call 的区别）

通过 apply 和 call 改变函数的 this 指向,他们两个函数的第一个参数都是一样的表示要改变指向的那个对象，第二个参数，apply 是数组，而 call 则是 arg1,arg2...这种形式。通过 bind 改变 this 作用域会返回一个新的函数，这个函数不会马上执行。

37.js 的各种位置，比如 clientHeight,scrollHeight,offsetHeight ,以及 scrollTop, offsetTop,clientTop 的区别？

clientHeight：表示的是可视区域的高度，不包含 border 和滚动条
offsetHeight：表示可视区域的高度，包含了 border 和滚动条
scrollHeight：表示了所有区域的高度，包含了因为滚动被隐藏的部分。
clientTop：表示边框 border 的厚度，在未指定的情况下一般为 0
scrollTop：滚动后被隐藏的高度，获取对象相对于由 offsetParent 属性指定的父坐标(css 定位的元素或 body 元素)距离顶端的高度。

38.js 拖拽功能的实现

首先是三个事件，分别是 mousedown，mousemove，mouseup
当鼠标点击按下的时候，需要一个 tag 标识此时已经按下，可以执行 mousemove 里面的具体方法。
clientX，clientY 标识的是鼠标的坐标，分别标识横坐标和纵坐标，并且我们用 offsetX 和 offsetY 来表示元素的元素的初始坐标，移动的举例应该是：
鼠标移动时候的坐标-鼠标按下去时候的坐标。
也就是说定位信息为：
鼠标移动时候的坐标-鼠标按下去时候的坐标+元素初始情况下的 offetLeft.
还有一点也是原理性的东西，也就是拖拽的同时是绝对定位，我们改变的是绝对定位条件下的 left
以及 top 等等值。
补充：也可以通过 html5 的拖放（Drag 和 drop）来实现

39.异步加载 js 的方法

defer：只支持 IE 如果您的脚本不会改变文档的内容，可将 defer 属性加入到<script>标签中，以便加快处理文档的速度。因为浏览器知道它将能够安全地读取文档的剩余部分而不用执行脚本，它将推迟对脚本的解释，直到文档已经显示给用户为止。
async，HTML5 属性仅适用于外部脚本，并且如果在 IE 中，同时存在 defer 和 async，那么 defer 的优先级比较高，脚本将在页面完成时执行。
创建 script 标签，插入到 DOM 中

40.Ajax 解决浏览器缓存问题

在 ajax 发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。
在 ajax 发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。
在 URL 后面加上一个随机数： "fresh=" + Math.random()。
在 URL 后面加上时间搓："nowtime=" + new Date().getTime()。
如果是使用 jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。这样页面的所有 ajax 都会执行这条语句就是不需要保存缓存记录。

41.js 的防抖

防抖（Debouncing）
防抖技术即是可以把多个顺序地调用合并成一次，也就是在一定时间内，规定事件被触发的次数。
通俗一点来说，看看下面这个简化的例子：

```
// 简单的防抖动函数
function debounce(func, wait, immediate) {
    // 定时器变量
    var timeout;
    return function() {
        // 每次触发 scroll handler 时先清除定时器
        clearTimeout(timeout);
        // 指定 xx ms 后触发真正想进行的操作 handler
        timeout = setTimeout(func, wait);
    };
};

// 实际想绑定在 scroll 事件上的 handler
function realFunc(){
    console.log("Success");
}

// 采用了防抖动
window.addEventListener('scroll',debounce(realFunc,500));
// 没采用防抖动
```

```
window.addEventListener('scroll',realFunc);
```

上面简单的防抖的例子可以拿到浏览器下试一下，大概功能就是如果 500ms 内没有连续触发两次 scroll 事件，那么才会触发我们真正想在 scroll 事件中触发的函数。
上面的示例可以更好的封装一下

```
// 防抖动函数
function debounce(func, wait, immediate) {
    var timeout;
    return function() {
        var context = this, args = arguments;
        var later = function() {
            timeout = null;
            if (!immediate) func.apply(context, args);
        };
        var callNow = immediate && !timeout;
        clearTimeout(timeout);
        timeout = setTimeout(later, wait);
        if (callNow) func.apply(context, args);
    };
};

var myEfficientFn = debounce(function() {
    // 滚动中的真正的操作
}, 250);

// 绑定监听
window.addEventListener('resize', myEfficientFn);
```

42.js 节流

防抖函数确实不错，但是也存在问题，譬如图片的懒加载，我希望在下滑过程中图片不断的被加载出来，而不是只有当我停止下滑时候，图片才被加载出来。又或者下滑时候的数据的 ajax 请求加载也是同理。
这个时候，我们希望即使页面在不断被滚动，但是滚动 handler 也可以以一定的频率被触发（譬如 250ms 触发一次），这类场景，就要用到另一种技巧，称为节流函数（throttling）。
节流函数，只允许一个函数在 X 毫秒内执行一次。
与防抖相比，节流函数最主要的不同在于它保证在 X 毫秒内至少执行一次我们希望触发的事件 handler。
与防抖相比，节流函数多了一个 mustRun 属性，代表 mustRun 毫秒内，必然会触发一次 handler ，同样是利用定时器，看看简单的示例：

```
// 简单的节流函数
function throttle(func, wait, mustRun) {
    var timeout,
        startTime = new Date();

    return function() {
        var context = this,
            args = arguments,
            curTime = new Date();

        clearTimeout(timeout);
        // 如果达到了规定的触发时间间隔，触发 handler
        if(curTime - startTime >= mustRun){
            func.apply(context,args);
            startTime = curTime;
        // 没达到触发间隔，重新设定定时器
        }else{
            timeout = setTimeout(func, wait);
        }
    };
};
// 实际想绑定在 scroll 事件上的 handler
function realFunc(){
    console.log("Success");
}
// 采用了节流函数
window.addEventListener('scroll',throttle(realFunc,500,1000));
```

上面简单的节流函数的例子可以拿到浏览器下试一下，大概功能就是如果在一段时间内 scroll 触发的间隔一直短于 500ms ，那么能保证事件我们希望调用的 handler 至少在 1000ms 内会触发一次。

43.JS 中的垃圾回收机制

必要性：由于字符串、对象和数组没有固定大小，所有当他们的大小已知时，才能对他们进行动态的存储分配。JavaScript 程序每次创建字符串、数组或对象时，解释器都必须分配内存来存储那个实体。只要像这样动态地分配了内存，最终都要释放这些内存以便他们能够被再用，否则，JavaScript 的解释器将会消耗完系统中所有可用的内存，造成系统崩溃。
这段话解释了为什么需要系统需要垃圾回收，JS 不像 C/C++，他有自己的一套垃圾回收机制（Garbage Collection）。JavaScript 的解释器可以检测到何时程序不再使用一个对象了，当他确定了一个对象是无用的时候，他就知道不再需要这个对象，可以把它所占用的内存释放掉了。例如：

```
var a="hello world";
var b="world";
var a=b;
//这时，会释放掉"hello world"，释放内存以便再引用
```
垃圾回收的方法：标记清除、计数引用。
标记清除
这是最常见的垃圾回收方式，当变量进入环境时，就标记这个变量为”进入环境“,从逻辑上讲，永远不能释放进入环境的变量所占的内存，永远不能释放进入环境变量所占用的内存，只要执行流程进入相应的环境，就可能用到他们。当离开环境时，就标记为离开环境。
垃圾回收器在运行的时候会给存储在内存中的变量都加上标记（所有都加），然后去掉环境变量中的变量，以及被环境变量中的变量所引用的变量（条件性去除标记），删除所有被标记的变量，删除的变量无法在环境变量中被访问所以会被删除，最后垃圾回收器，完成了内存的清除工作，并回收他们所占用的内存。
引用计数法
另一种不太常见的方法就是引用计数法，引用计数法的意思就是每个值没引用的次数，当声明了一个变量，并用一个引用类型的值赋值给改变量，则这个值的引用次数为 1,；相反的，如果包含了对这个值引用的变量又取得了另外一个值，则原先的引用值引用次数就减 1，当这个值的引用次数为 0 的时候，说明没有办法再访问这个值了，因此就把所占的内存给回收进来，这样垃圾收集器再次运行的时候，就会释放引用次数为 0 的这些值。
用引用计数法会存在内存泄露，下面来看原因：
```
function problem() {
var objA = new Object();
var objB = new Object();
objA.someOtherObject = objB;
objB.anotherObject = objA;
}
```
在这个例子里面，objA 和 objB 通过各自的属性相互引用，这样的话，两个对象的引用次数都为 2，在采用引用计数的策略中，由于函数执行之后，这两个对象都离开了作用域，函数执行完成之后，因为计数不为 0，这样的相互引用如果大量存在就会导致内存泄露。
特别是在 DOM 对象中，也容易存在这种问题：
```
var element=document.getElementById（’ ‘);
var myObj=new Object();
myObj.element=element;
element.someObject=myObj;
```
这样就不会有垃圾回收的过程。

44.eval 是做什么的

它的功能是将对应的字符串解析成 js 并执行，应该避免使用 js，因为非常消耗

性能（2 次，一次解析成 js，一次执行）

45. 如何理解前端模块化

前端模块化就是复杂的文件编程一个一个独立的模块，比如 js 文件等等，分成独立的模块有利于重用（复用性）和维护（版本迭代），这样会引来模块之间相互依赖的问题，所以有了 commonJS 规范，AMD，CMD 规范等等，以及用于 js 打包（编译等处理）的工具 webpack

46.说一下 Commonjs、AMD 和 CMD

一个模块是能实现特定功能的文件，有了模块就可以方便的使用别人的代码，想要什么功能就能加载什么模块。
Commonjs：开始于服务器端的模块化，同步定义的模块化，每个模块都是一个单独的作用域，模块输出，modules.exports，模块加载 require()引入模块。
AMD：中文名异步模块定义的意思。
requireJS 实现了 AMD 规范，主要用于解决下述两个问题。
1.多个文件有依赖关系，被依赖的文件需要早于依赖它的文件加载到浏览器
2.加载的时候浏览器会停止页面渲染,加载文件越多,页面失去响应的时间越长。
语法：requireJS 定义了一个函数 define，它是全局变量，用来定义模块。
requireJS 的例子：

```
//定义模块
define(['dependency'], function(){
var name = 'Byron';
function printName(){
console.log(name);
}
return {
printName: printName
};
});
//加载模块
require(['myModule'], function (my){
my.printName();
}
```

requirejs 定义了一个函数 define,它是全局变量，用来定义模块：

```
define(id?dependencies?,factory)
```

在页面上使用模块加载函数：

require([dependencies],factory);
总结 AMD 规范：require（）函数在加载依赖函数的时候是异步加载的，这样浏览器不会失去响应，它指定的回调函数，只有前面的模块加载成功，才会去执行。因为网页在加载 js 的时候会停止渲染，因此我们可以通过异步的方式去加载 js，而如果需要依赖某些，也是异步去依赖，依赖后再执行某些方法。

46.对象深度克隆的简单实现

```
function deepClone(obj){
var newObj= obj instanceof Array ? []:{};
for(var item in obj){
var temple= typeof obj[item] == 'object' ? deepClone(obj[item]):obj[item];
newObj[item] = temple;
}
return newObj;
}
```

ES5 的常用的对象克隆的一种方式。注意数组是对象，但是跟对象又有一定区别，所以我们一开始判断了一些类型，决定 newObj 是对象还是数组~

47.实现一个 once 函数，传入函数参数只执行一次

```
function ones(func){
var tag=true;
return function(){
if(tag==true){
func.apply(null,arguments);
tag=false;
}
return undefined
}
}
```

48.将原生的 ajax 封装成 promise

```
var  myNewAjax=function(url){
return new Promise(function(resolve,reject){
```

```
var xhr = new XMLHttpRequest();
xhr.open('get',url);
xhr.send(data);
xhr.onreadystatechange=function(){
if(xhr.status==200&&readyState==4){
var json=JSON.parse(xhr.responseText);
resolve(json)
}else if(xhr.readyState==4&&xhr.status!=200){
reject('error');
}
}
})
}
```

49.js 监听对象属性的改变

我们假设这里有一个 user 对象,
(1)在 ES5 中可以通过 Object.defineProperty 来实现已有属性的监听

```
Object.defineProperty(user,'name',{
set：function(key,value){
}
})
```

缺点：如果 id 不在 user 对象中，则不能监听 id 的变化
(2)在 ES6 中可以通过 Proxy 来实现

```
var  user = new Proxy({}, {
set：function(target,key,value,receiver){
}
})
```

这样即使有属性在 user 中不存在，通过 user.id 来定义也同样可以这样监听这个属性的变化哦~

50.如何实现一个私有变量，用 getName 方法可以访问，不能直接访问

(1)通过 defineProperty 来实现

```
obj={
name:yuxiaoliang,
getName:function(){
```

```
return this.name
}
}
object.defineProperty(obj,"name",{
//不可枚举不可配置

});
```
(2)通过函数的创建形式
复制代码
```
function product(){
var name='yuxiaoliang';
this.getName=function(){
return name;
}
}
var obj=new product();
```

57、==和===、以及 Object.is 的区别

(1) ==
主要存在：强制转换成 number,null==undefined
```
" "==0  //true
"0"==0  //true
" " !="0" //true
123=="123" //true
null==undefined //true
```
(2)Object.js
主要的区别就是+0！=-0 而 NaN==NaN
(相对比===和==的改进)

51.setTimeout、setInterval 和 requestAnimationFrame 之间的区别

与 setTimeout 和 setInterval 不同，requestAnimationFrame 不需要设置时间间隔，
大多数电脑显示器的刷新频率是 60Hz，大概相当于每秒钟重绘 60 次。大多数浏览器都会对重绘操作加以限制，不超过显示器的重绘频率，因为即使超过那个频率用户体验也不会有提升。因此，最平滑动画的最佳循环间隔是 1000ms/60，约等于 16.6ms。
RAF 采用的是系统时间间隔，不会因为前面的任务，不会影响 RAF，但是如果前面的任务多的话，

会响应 setTimeout 和 setInterval 真正运行时的时间间隔。
特点：
（1）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。
（2）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
（3）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。

52.实现一个两列等高布局，讲讲思路

为了实现两列等高，可以给每列加上 padding-bottom:9999px;
margin-bottom:-9999px;同时父元素设置 overflow:hidden;

53.自己实现一个 bind 函数

原理：通过 apply 或者 call 方法来实现。

(1)初始版本
```
Function.prototype.bind=function(obj,arg){
var arg=Array.prototype.slice.call(arguments,1);
var context=this;
return function(newArg){
arg=arg.concat(Array.prototype.slice.call(newArg));
return context.apply(obj,arg);
}
}
```
(2) 考虑到原型链
为什么要考虑？因为在 new 一个 bind 过生成的新函数的时候，必须的条件是要继承原函数的原型
```
Function.prototype.bind=function(obj,arg){
var arg=Array.prototype.slice.call(arguments,1);
var context=this;
var bound=function(newArg){
arg=arg.concat(Array.prototype.slice.call(newArg));
return context.apply(obj,arg);
}
var F=function(){}
```

```
//这里需要一个寄生组合继承
F.prototype=context.prototype;
bound.prototype=new F();
return bound;
}
```

54.用 setTimeout()方法来模拟 setInterval()与 setInterval()之间的什么区别？

首先来看 setInterval 的缺陷，使用 setInterval()创建的定时器确保了定时器代码规则地插入队列中。这个问题在于：如果定时器代码在代码再次添加到队列之前还没完成执行,结果就会导致定时器代码连续运行好几次。而之间没有间隔。不过幸运的是：javascript 引擎足够聪明，能够避免这个问题。当且仅当没有该定时器的如何代码实例时，才会将定时器代码添加到队列中。这确保了定时器代码加入队列中最小的时间间隔为指定时间。
这种重复定时器的规则有两个问题：1.某些间隔会被跳过 2.多个定时器的代码执行时间可能会比预期小。

55. js 怎么控制一次加载一张图片，加载完后再加载下一张

方法 1

```
<script type="text/javascript">
var obj=new Image();
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";
obj.onload=function(){
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);

document.getElementById("mypic").innnerHTML="<img  src='"+this.src+"' />";
}
</script>
<div id="mypic">onloading……</div>
```

(2)方法 2

```
<script type="text/javascript">
var obj=new Image();
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";
```

```
obj.onreadystatechange=function(){
if(this.readyState=="complete"){
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);
document.getElementById("mypic").innnerHTML="<img  src='"+this.src+"'
/>";
}
}
</script>
<div id="mypic">onloading……</div>
```

56.如何实现 sleep 的效果（es5 或者 es6）

(1)while 循环的方式

```
function sleep(ms){
var start=Date.now(),expire=start+ms;
while(Date.now()<expire);
console.log('1111');
return;
}
```

执行 sleep(1000)之后，休眠了 1000ms 之后输出了 1111。上述循环的方式缺点很明显，容易造成死循环。

(2)通过 promise 来实现

```
function sleep(ms){
var temple=new Promise(
(resolve)=>{
console.log(111);setTimeout(resolve,ms)
});
return temple
}
sleep(500).then(function(){
//console.log(222)
})
```

//先输出了 111，延迟 500ms 后输出 222

(3)通过 async 封装

```
function sleep(ms){
return new Promise((resolve)=>setTimeout(resolve,ms));
}
async function test(){
var temple=await sleep(1000);
console.log(1111)
return temple
}
```

```
test();
//延迟 1000ms 输出了 1111
(4).通过 generate 来实现
function* sleep(ms){
yield new Promise(function(resolve,reject){
console.log(111);
setTimeout(resolve,ms);
})
}
sleep(500).next().value.then(function(){console.log(2222)})
```

57.Function._proto_(getPrototypeOf)是什么？

获取一个对象的原型，在 chrome 中可以通过_proto_的形式，或者在 ES6 中可以通过 Object.getPrototypeOf 的形式。
那么 Function.proto 是什么么？ 也就是说 Function 由什么对象继承而来，我们来做如下判别。

```
Function.__proto__==Object.prototype //false
Function.__proto__==Function.prototype//true
```

我们发现 Function 的原型也是 Function。

58.实现 js 中所有对象的深度克隆（包装对象，Date 对象，正则对象）

通过递归可以简单实现对象的深度克隆，但是这种方法不管是 ES6 还是 ES5 实现，都有同样的缺陷，就是只能实现特定的 object 的深度复制（比如数组和函数），不能实现包装对象 Number，String ， Boolean，以及 Date 对象，RegExp 对象的复制。
(1)前文的方法

```
function deepClone(obj){
var newObj= obj instanceof Array?[]:{};
for(var i in obj){
newObj[i]=typeof obj[i]=='object'?
deepClone(obj[i]):obj[i];
}
return newObj;
}
```

这种方法可以实现一般对象和数组对象的克隆，比如：

```
var arr=[1,2,3];
var newArr=deepClone(arr);
// newArr->[1,2,3]
var obj={
x:1,
y:2
}
var newObj=deepClone(obj);
// newObj={x:1,y:2}
```

但是不能实现例如包装对象 Number,String,Boolean,以及正则对象 RegExp 和 Date 对象的克隆，比如：

```
//Number 包装对象

var num=new Number(1);
typeof num // "object"
var newNum=deepClone(num);
//newNum ->  {} 空对象


//String 包装对象

var str=new String("hello");
typeof str //"object"
var newStr=deepClone(str);
//newStr->  {0:'h',1:'e',2:'l',3:'l',4:'o'};

//Boolean 包装对象

var bol=new Boolean(true);
typeof bol //"object"
var newBol=deepClone(bol);
// newBol ->{} 空对象


....
```

(2)valueof()函数

所有对象都有 valueOf 方法，valueOf 方法对于：如果存在任意原始值，它就默认将对象转换为表示它的原始值。对象是复合值，而且大多数对象无法真正表示为一个原始值，因此默认的 valueOf()方法简单地返回对象本身，而不是返回一个原始值。数组、函数和正则表达式简单地继承了这个默认方法，调用这些类型的实例的 valueOf()方法只是简单返回这个对象本身。

对于原始值或者包装类：

```
function baseClone(base){
return base.valueOf();
}
//Number
var num=new Number(1);
var newNum=baseClone(num);
//newNum->1
//String
var str=new String('hello');
var newStr=baseClone(str);
// newStr->"hello"
//Boolean
var bol=new Boolean(true);
var newBol=baseClone(bol);
//newBol-> true
```

其实对于包装类，完全可以用=号来进行克隆，其实没有深度克隆一说，
这里用 valueOf 实现，语法上比较符合规范。
对于 Date 类型：
因为 valueOf 方法,日期类定义的 valueOf()方法会返回它的一个内部表示:1970 年 1 月 1 日以来的毫秒数.因此我们可以在 Date 的原型上定义克隆的方法：

```
Date.prototype.clone=function(){
return new Date(this.valueOf());
}
var date=new Date('2010');
var newDate=date.clone();
// newDate->  Fri Jan 01 2010 08:00:00 GMT+0800
```

对于正则对象 RegExp：

```
RegExp.prototype.clone = function() {
var pattern = this.valueOf();
var flags = '';
flags += pattern.global ? 'g' : '';
flags += pattern.ignoreCase ? 'i' : '';
flags += pattern.multiline ? 'm' : '';
return new RegExp(pattern.source, flags);
};
var reg=new RegExp('/111/');
var newReg=reg.clone();
//newReg->  /\/111\//
```

59.简单实现 Node 的 Events 模块

简介：观察者模式或者说订阅模式，它定义了对象间的一种一对多的关系，让多个观察者对象同时监听某一个主题对象，当一个对象发生改变时，所有依赖于它的对象都将得到通知。
node 中的 Events 模块就是通过观察者模式来实现的：

```
var events=require('events');
var eventEmitter=new events.EventEmitter();
eventEmitter.on('say',function(name){
console.log('Hello',name);
})
eventEmitter.emit('say','Jony yu');
```

这样，eventEmitter 发出 say 事件，通过 On 接收，并且输出结果，这就是一个订阅模式的实现，下面我们来简单的实现一个 Events 模块的 EventEmitter。
(1)实现简单的 Event 模块的 emit 和 on 方法

```
function Events(){
this.on=function(eventName,callBack){
if(!this.handles){
this.handles={};
}
if(!this.handles[eventName]){
this.handles[eventName]=[];
}
this.handles[eventName].push(callBack);
}
this.emit=function(eventName,obj){
if(this.handles[eventName]){
for(var i=0;o<this.handles[eventName].length;i++){
this.handles[eventName][i](obj);
}
}
}
return this;
}
```

这样我们就定义了 Events，现在我们可以开始来调用：

```
var events=new Events();
events.on('say',function(name){
console.log('Hello',nama)
});
events.emit('say','Jony yu');
//结果就是通过 emit 调用之后，输出了 Jony yu
```

(2)每个对象是独立的

因为是通过 new 的方式，每次生成的对象都是不相同的，因此：

```
var event1=new Events();
```

```
var event2=new Events();
event1.on('say',function(){
console.log('Jony event1');
});
event2.on('say',function(){
console.log('Jony event2');
})
event1.emit('say');
event2.emit('say');
//event1、event2 之间的事件监听互相不影响

//输出结果为'Jony event1' 'Jony event2'
```

60.箭头函数中 this 指向举例

```
var a=11;
function test2(){
this.a=22;
let b=()=>{console.log(this.a)}
b();
}
var x=new test2();
//输出 22
```

定义时绑定。

61. js 判断类型

判断方法：typeof()， instanceof，Object.prototype.toString.call()等

62.数组常用方法

push()，pop()，shift()，unshift()，splice()，sort()，reverse()，map()等

63.数组去重

法一：indexOf 循环去重
法二：ES6 Set 去重；Array.from(new Set(array))
法三：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

64.闭包有什么用

（1）什么是闭包：
闭包是指有权访问另外一个函数作用域中的变量的函数。
闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。
（2）为什么要用：
匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。
结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。
封装：实现类和继承等。

65.事件代理在捕获阶段的实际应用

可以在父元素层面阻止事件向子元素传播，也可代替子元素执行某些操作。

66.去除字符串首尾空格

使用正则(^\s*)|(\s*$)即可

67.性能优化

减少 HTTP 请求
使用内容发布网络（CDN）
添加本地缓存
压缩资源文件
将 CSS 样式表放在顶部，把 javascript 放在底部（浏览器的运行机制决定）
避免使用 CSS 表达式
减少 DNS 查询
使用外部 javascript 和 CSS
避免重定向
图片 lazyLoad

68.来讲讲 JS 的闭包吧

闭包是指有权访问另外一个函数作用域中的变量的函数。
闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。
（2）为什么要用：
匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。
结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

69.能来讲讲 JS 的语言特性吗

运行在客户端浏览器上；
不用预编译，直接解析执行代码；
是弱类型语言，较为灵活；
与操作系统无关，跨平台的语言；
脚本语言、解释性语言

70.如何判断一个数组(讲到 typeof 差点掉坑里)

Object.prototype.call.toString()
Instanceof

71.你说到 typeof，能不能加一个限制条件达到判断条件

typeof 只能判断是 object,可以判断一下是否拥有数组的方法

72.JS 实现跨域

JSONP：通过动态创建 script，再请求一个带参网址实现跨域通信。
document.domain + iframe 跨域：两个页面都通过 js 强制设置 document.domain 为基础主域，就实现了同域。
location.hash + iframe 跨域：a 欲与 b 跨域相互通信，通过中间页 c 来实现。三个页面，不同域之间利用 iframe 的 location.hash 传值，相同域之间直接 js 访问来通信。

window.name + iframe 跨域：通过 iframe 的 src 属性由外域转向本地域，跨域数据即由 iframe 的 window.name 从外域传递到本地域。
postMessage 跨域：可以跨域操作的 window 属性之一。
CORS：服务端设置 Access-Control-Allow-Origin 即可，前端无须设置，若要带 cookie 请求，前后端都需要设置。
代理跨域：启一个代理服务器，实现数据的转发

73.Js 基本数据类型

基本数据类型：undefined、null、number、boolean、string、symbol

74.js 深度拷贝一个元素的具体实现

```
var deepCopy = function(obj) {
if (typeof obj !== 'object') return;
var newObj = obj instanceof Array ? [] : {};
for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) :
obj[key];
}
}
return newObj;
}
```

75.之前说了 ES6set 可以数组去重，是否还有数组去重的方法

法一：indexOf 循环去重
法二：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

76.重排和重绘，讲讲看

重绘（repaint 或 redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。
触发重绘的条件：改变元素外观属性。如：color，background-color 等。
注意：table 及其内部元素可能需要多次计算才能确定好其在渲染树中节点的属

性值，比同等元素要多花两倍时间，这就是我们尽量避免使用 table 布局页面的原因之一。
重排（重构/回流/reflow）：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。
重绘和重排的关系：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，该过程称为重绘。所以，重排必定会引发重绘，但重绘不一定会引发重排。

77.JS 的全排列

```
function permutate(str) {
var result = [];
if(str.length > 1) {
var left = str[0];
var rest = str.slice(1, str.length);
var preResult = permutate(rest);
for(var i=0; i<preResult.length; i++) {
for(var j=0; j<preResult[i].length; j++) {
var tmp = preResult[i],slice(0, j) + left + preResult[i].slice(j,
preResult[i].length);
result.push(tmp);
}
}
} else if (str.length == 1) {
return [str];
}
return result;
}
```

78.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，是浏览器对 JavaScript 实施的安全限制，那么只要协议、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。

79.不同数据类型的值的比较，是怎么转换的，有什么规则

比较运算x==y, 其中x和 y是值，产生true或者false。这样的比较按如下方式进行：

1. 若Type(x)与Type(y)相同， 则
    a. 若Type(x)为Undefined， 返回true。
    b. 若Type(x)为Null， 返回true。
    c. 若Type(x)为Number， 则
        i. 若x为NaN， 返回false。
        ii. 若y为NaN， 返回false。
        iii. 若x与y为相等数值， 返回true。
        iv. 若x 为 +0 且 y为-0， 返回true。
        v. 若x 为 -0 且 y为+0， 返回true。
        vi. 返回false。
    d. 若Type(x)为String, 则当x和y为完全相同的字符序列（长度相等且相同字符在相同位置）时返回true。 否则，返回false。
    e. 若Type(x)为Boolean, 当x和y为同为true或者同为false时返回true。 否则， 返回false。
    f. 当x和y为引用同一对象时返回true。否则，返回false。
2. 若x为null且y为undefined， 返回true。
3. 若x为undefined且y为null， 返回true。
4. 若Type(x) 为 Number 且 Type(y)为String， 返回comparison x == ToNumber(y)的结果。
5. 若Type(x) 为 String 且 Type(y)为Number，
6. 返回比较ToNumber(x) == y的结果。
7. 若Type(x)为Boolean， 返回比较ToNumber(x) == y的结果。
8. 若Type(y)为Boolean， 返回比较x == ToNumber(y)的结果。
9. 若Type(x)为String或Number，且Type(y)为Object，返回比较x == ToPrimitive(y)的结果。
10. 若Type(x)为Object且Type(y)为String或Number， 返回比较ToPrimitive(x) == y的结果。
11. 返回false。

80、null == undefined 为什么

要比较相等性之前，不能将 null 和 undefined 转换成其他任何值，但 null == undefined 会返回 true 。ECMAScript 规范中是这样定义的。

81、this 的指向 哪几种

默认绑定：全局环境中，this 默认绑定到 window。
隐式绑定：一般地，被直接对象所包含的函数调用时，也称为方法调用，this 隐式绑定到该直接对象。
隐式丢失：隐式丢失是指被隐式绑定的函数丢失绑定对象，从而默认绑定到 window。显式绑定：通过 call()、apply()、bind()方法把对象绑定到 this 上，叫做显式绑定。
new 绑定：如果函数或者方法调用之前带有关键字 new，它就构成构造函数调用。对于 this 绑定来说，称为 new 绑定。

【1】构造函数通常不使用 return 关键字，它们通常初始化新对象，当构造函数的函数体执行完毕时，它会显式返回。在这种情况下，构造函数调用表达式的计算结果就是这个新对象的值。
【2】如果构造函数使用 return 语句但没有指定返回值，或者返回一个原始值，那么这时将忽略返回值，同时使用这个新对象作为调用结果。
【3】如果构造函数显式地使用 return 语句返回一个对象，那么调用表达式的值就是这个对象。

89、暂停死区

在代码块内，使用 let、const 命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”

82、AngularJS 双向绑定原理

Angular 将双向绑定转换为一堆 watch 表达式，然后递归这些表达式检查是否发生过变化，如果变了则执行相应的 watcher 函数（指 view 上的指令，如 ng-bind，ng-show 等或是{{}}）。等到 model 中的值不再发生变化，也就不会再有 watcher 被触发，一个完整的 digest 循环就完成了。
Angular 中在 view 上声明的事件指令，如：ng-click、ng-change 等，会将浏览器的事件转发给$scope 上相应的 model 的响应函数。等待相应函数改变 model，紧接着触发脏检查机制刷新 view。

watch 表达式：可以是一个函数、可以是$scope 上的一个属性名，也可以是一个字符串形式的表达式。$watch 函数所监听的对象叫做 watch 表达式。watcher 函数：指在 view 上的指令（ngBind，ngShow、ngHide 等）以及{{}}表达式，他们所注册的函数。每一个 watcher 对象都包括：监听函数，上次变化的值，获取监听表达式的方法以及监听表达式，最后还包括是否需要使用深度对比（angular.equals()）

83、写一个深度拷贝

```
function clone( obj ) {
var copy;
switch( typeof obj ) {
case "undefined":
```

```
break;
case "number":
copy = obj - 0;
break;
case "string":
copy = obj + "";
break;
case "boolean":
copy = obj;
break;
case "object":  //object 分为两种情况 对象（Object）和数组（Array）

if(obj === null) {
copy = null;
} else {
if( Object.prototype.toString.call(obj).slice(8, -1) === "Array") {
copy = [];
for( var i = 0 ; i < obj.length ; i++ ) {
copy.push(clone(obj[i]));
}
} else {
copy = {};
for( var j in obj) {
copy[j] = clone(obj[j]);
}
}
}
break;
default:
copy = obj;
break;
}
return copy;
}
```

84、简历中提到了 requestAnimationFrame，请问是怎么使用的

requestAnimationFrame() 方法告诉浏览器您希望执行动画并请求浏览器在下一次重绘之前调用指定的函数来更新动画。该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

85、有一个游戏叫做 Flappy Bird，就是一只小鸟在飞，前面是无尽的沙漠，上下不断有钢管生成，你要躲避钢管。然后小明在玩这个游戏时候老是卡顿甚至崩溃，说出原因（3-5 个）以及解决办法（3-5 个）

原因可能是：
1.内存溢出问题。
2.资源过大问题。
3.资源加载问题。
4.canvas 绘制频率问题
解决办法：
1.针对内存溢出问题，我们应该在钢管离开可视区域后，销毁钢管，让垃圾收集器回收钢管，因为不断生成的钢管不及时清理容易导致内存溢出游戏崩溃。

2.针对资源过大问题，我们应该选择图片文件大小更小的图片格式，比如使用 webp、png 格式的图片，因为绘制图片需要较大计算量。
3.针对资源加载问题，我们应该在可视区域之前就预加载好资源，如果在可视区域生成钢管的话，用户的体验就认为钢管是卡顿后才生成的，不流畅。
4.针对 canvas 绘制频率问题，我们应该需要知道大部分显示器刷新频率为 60 次/s,因此游戏的每一帧绘制间隔时间需要小于 1000/60=16.7ms，才能让用户觉得不卡顿。
（注意因为这是单机游戏，所以回答与网络无关）

86、 什么是按需加载

当用户触发了动作时才加载对应的功能。触发的动作，是要看具体的业务场景而言，包括但不限于以下几个情况：鼠标点击、输入文字、拉动滚动条，鼠标移动、窗口大小更改等。加载的文件，可以是 JS、图片、CSS、HTML 等。

87、说一下什么是 virtual dom

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的

DOM 树，插到文档当中 当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异 把所记录的差异应用到所构建的真正的 DOM 树上，视图就更新了。Virtual DOM 本质上就是在 JS 和 DOM 之间做了一个缓存

88、webpack 用来干什么的

webpack 是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

89、ant-design 优点和缺点

优点：组件非常全面，样式效果也都比较不错。
缺点：框架自定义程度低，默认 UI 风格修改困难

90、JS 中继承实现的几种方式

1、原型链继承，将父类的实例作为子类的原型，他的特点是实例是子类的实例也是父类的实例，父类新增的原型方法/属性，子类都能够访问，并且原型链继承简单易于实现，缺点是来自原型对象的所有属性被所有实例共享，无法实现多继承，无法向父类构造函数传参。
2、构造继承，使用父类的构造函数来增强子类实例，即复制父类的实例属性给子类，

构造继承可以向父类传递参数，可以实现多继承，通过 call 多个父类对象。但是构造继承只能继承父类的实例属性和方法，不能继承原型属性和方法，无法实现函数服用，每个子类都有父类实例函数的副本，影响性能
3、实例继承，为父类实例添加新特性，作为子类实例返回，实例继承的特点是不限制调用方法，不管是 new 子类（）还是子类（）返回的对象具有相同的效果，缺点是实例是父类的实例，不是子类的实例，不支持多继承
4、拷贝继承：特点：支持多继承，缺点：效率较低，内存占用高（因为要拷贝父类的属性）无法获取父类不可枚举的方法（不可枚举方法，不能使用 for in 访问到）
5、组合继承：通过调用父类构造，继承父类的属性并保留传参的优点，然后通

过将父类实例作为子类原型，实现函数复用
6、寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免的组合继承的缺点

91、写一个函数，第一秒打印 1，第二秒打印 2

两个方法，第一个是用 let 块级作用域

```
for(let i=0;i<5;i++){
setTimeout(function(){
console.log(i)
},1000*i)
}
```

第二个方法闭包

```
for(var i=0;i<5;i++){
(function(i){
setTimeout(function(){
console.log(i)
},1000*i)
})(i)
}
```

92、vue 的生命周期

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、编译模板、挂载 Dom、渲染→更新→渲染、销毁等一系列过程，我们称这是 Vue 的生命周期。通俗说就是 Vue 实例从创建到销毁的过程，就是生命周期。
每一个组件或者实例都会经历一个完整的生命周期，总共分为三个阶段：初始化、运行中、销毁。
实例、组件通过 new Vue() 创建出来之后会初始化事件和生命周期，然后就会执行 beforeCreate 钩子函数，这个时候，数据还没有挂载呢，只是一个空壳，无法访问到数据和真实的 dom，一般不做操作

挂载数据，绑定事件等等，然后执行 created 函数，这个时候已经可以使用到数据，也可以更改数据,在这里更改数据不会触发 updated 函数，在这里可以在渲染前倒数第二次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取
接下来开始找实例或者组件对应的模板，编译模板为虚拟 dom 放入到 render 函

数中准备渲染，然后执行 beforeMount 钩子函数，在这个函数中虚拟 dom 已经创建完成，马上就要渲染,在这里也可以更改数据，不会触发 updated，在这里可以在渲染前最后一次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取
接下来开始 render，渲染出真实 dom，然后执行 mounted 钩子函数，此时，组件已经出现在页面中，数据、真实 dom 都已经处理好了,事件都已经挂载好了，可以在这里操作真实 dom 等事情...
当组件或实例的数据更改之后，会立即执行 beforeUpdate，然后 vue 的虚拟 dom 机制会重新构建虚拟 dom 与上一次的虚拟 dom 树利用 diff 算法进行对比之后重新渲染，一般不做什么事儿
当更新完成后，执行 updated，数据已经更改完成，dom 也重新 render 完成，可以操作更新后的虚拟 dom
当经过某种途径调用$destroy 方法后，立即执行 beforeDestroy，一般在这里做一些善后工作，例如清除计时器、清除非指令绑定的事件等等
组件的数据绑定、监听...去掉后只剩下 dom 空壳，这个时候，执行 destroyed，在这里做善后工作也可以

93、简单介绍一下 symbol

Symbol 是 ES6 的新增属性，代表用给定名称作为唯一标识，这种类型的值可以这样创建，let id=symbol(“id”)
Symbl 确保唯一，即使采用相同的名称，也会产生不同的值，我们创建一个字段，仅为知道对应 symbol 的人能访问，使用 symbol 很有用，symbol 并不是 100%隐藏，有内置方法 Object.getOwnPropertySymbols(obj)可以获得所有的 symbol。也有一个方法 Reflect.ownKeys(obj)返回对象所有的键，包括 symbol。
所以并不是真正隐藏。但大多数库内置方法和语法结构遵循通用约定他们是隐藏的。

94、什么是事件监听

addEventListener()方法，用于向指定元素添加事件句柄，它可以更简单的控制事件，语法为
element.addEventListener(event, function, useCapture);
第一个参数是事件的类型(如 "click" 或 "mousedown").
第二个参数是事件触发后调用的函数。
第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。
事件传递有两种方式，冒泡和捕获
事件传递定义了元素事件触发的顺序，如果你将 P 元素插入到 div 元素中，用户点击 P 元素，

在冒泡中，内部元素先被触发，然后再触发外部元素，
捕获中，外部元素先被触发，在触发内部元素。

95、介绍一下 promise，及其底层如何实现

Promise 是一个对象，保存着未来将要结束的事件，她有两个特征:
1、对象的状态不受外部影响，Promise 对象代表一个异步操作，有三种状态，pending 进行中，fulfilled 已成功，rejected 已失败，只有异步操作的结果，才可以决定当前是哪一种状态，任何其他操作都无法改变这个状态，这也就是 promise 名字的由来
2、一旦状态改变，就不会再变，promise 对象状态改变只有两种可能，从 pending 改到 fulfilled 或者从 pending 改到 rejected，只要这两种情况发生，状态就凝固了，不会再改变，这个时候就称为定型 resolved,
Promise 的基本用法，

```
let promise1 = new Promise(function(resolve,reject){
setTimeout(function(){
resolve('ok')
},1000)
})
promise1.then(function success(val){
console.log(val)
})
```

最简单代码实现 promise

```
class PromiseM {
constructor (process) {
this.status = 'pending'
this.msg = ''
process(this.resolve.bind(this), this.reject.bind(this))
return this
}
resolve (val) {
this.status = 'fulfilled'
this.msg = val
}
reject (err) {
this.status = 'rejected'
this.msg = err
}
then (fufilled, reject) {
if(this.status === 'fulfilled') {
fufilled(this.msg)
}
```

```
if(this.status === 'rejected') {
reject(this.msg)
}
}
}
//测试代码
var mm=new PromiseM(function(resolve,reject){
resolve('123');
});
mm.then(function(success){
console.log(success);
},function(){
console.log('fail!');
});
```

95、说说 C++,Java，JavaScript 这三种语言的区别

从静态类型还是动态类型来看
静态类型,编译的时候就能够知道每个变量的类型,编程的时候也需要给定类型，如 Java 中的整型 int，浮点型 float 等。C、C++、Java 都属于静态类型语言。
动态类型，运行的时候才知道每个变量的类型，编程的时候无需显示指定类型，如 JavaScript 中的 var、PHP 中的$。JavaScript、Ruby、Python 都属于动态类型语言。
静态类型还是动态类型对语言的性能有很大影响。
对于静态类型，在编译后会大量利用已知类型的优势，如 int 类型，占用 4 个字节，编译后的代码就可以用内存地址加偏移量的方法存取变量，而地址加偏移量的算法汇编很容易实现。
对于动态类型，会当做字符串通通存下来，之后存取就用字符串匹配。
从编译型还是解释型来看
编译型语言，像 C、C++，需要编译器编译成本地可执行程序后才能运行，由开发人员在编写完成后手动实施。用户只使用这些编译好的本地代码，这些本地代码由系统加载器执行，由操作系统的 CPU 直接执行，无需其他额外的虚拟机等。
源代码=》抽象语法树=》中间表示=》本地代码
解释性语言，像 JavaScript、Python，开发语言写好后直接将代码交给用户，用户使用脚本解释器将脚本文件解释执行。对于脚本语言，没有开发人员的编译过程，当然，也不绝对。
源代码=》抽象语法树=》解释器解释执行。
对于 JavaScript，随着 Java 虚拟机 JIT 技术的引入，工作方式也发生了改变。可以将抽象语法树转成中间表示（字节码），再转成本地代码，如 JavaScriptCore，这样可以大大提高执行效率。也可以从抽象语法树直接转成本地代码，如 V8
Java 语言，分为两个阶段。首先像 C++语言一样，经过编译器编译。和 C++的不

同，C++编译生成本地代码，Java 编译后，生成字节码，字节码与平台无关。第二阶段，由 Java 的运行环境也就是 Java 虚拟机运行字节码，使用解释器执行这些代码。一般情况下，Java 虚拟机都引入了 JIT 技术，将字节码转换成本地代码来提高执行效率。
注意，在上述情况中，编译器的编译过程没有时间要求，所以编译器可以做大量的代码优化措施。
对于 JavaScript 与 Java 它们还有的不同：
对于 Java，Java 语言将源代码编译成字节码，这个同执行阶段是分开的。也就是从源代码到抽象语法树到字节码这段时间的长短是无所谓的。
对于 JavaScript，这些都是在网页和 JavaScript 文件下载后同执行阶段一起在网页的加载和渲染过程中实施的，所以对于它们的处理时间有严格要求。

96、js 原型链，原型链的顶端是什么？Object 的原型是什么？Object 的原型的原型是什么？在数组原型链上实现删除数组重复数据的方法

能够把这个讲清楚弄明白是一件很困难的事，
首先明白原型是什么，在 ES6 之前，JS 没有类和继承的概念，JS 是通过原型来实现继承的，在 JS 中一个构造函数默认带有一个 prototype 属性，这个的属性值是一个对象，同时这个 prototype 对象自带有一个 constructor 属性，这个属性指向这个构造函数，同时每一个实例都会有一个_proto_属性指向这个 prototype 对象，我们可以把这个叫做隐式原型，我们在使用一个实例的方法的时候，会先检查这个实例中是否有这个方法，没有的话就会检查这个 prototype 对象是否有这个方法，
基于这个规则，如果让原型对象指向另一个类型的实例，即 constructor1.protoytpe=instance2，这时候如果试图引用 constructor1 构造的实例 instance1 的某个属性 p1，
首先会在 instance1 内部属性中找一遍，
接着会在 instance1._proto_（constructor1.prototype）即是 instance2 中寻找 p1

搜寻轨迹：instance1->instance2->constructor2.prototype……->Object.prototype;这即是原型链，原型链顶端是 Object.prototype

97、什么是 JavaScript

JavaScript 一种动态类型、弱类型、基于原型的客户端脚本语言，用来给 HTML 网页增加动态功能。
动态：
在运行时确定数据类型。变量使用之前不需要类型声明，通常变量的类型是被赋值的那个值的类型。
弱类：
计算时可以不同类型之间对使用者透明地隐式转换，即使类型不正确，也能通过隐式转换来得到正确的类型。
原型：
新对象继承对象（作为模版），将自身的属性共享给新对象，模版对象称为原型。这样新对象实例化后不但可以享有自己创建时和运行时定义的属性，而且可以享有原型对象的属性。

98、JavaScript 组成部分：

ECMAScript（核心）
作为核心，它规定了语言的组成部分：语法、类型、语句、关键字、保留字、操作符、对象。
DOM（文档对象模型）
DOM 把整个页面映射为一个多层节点结果，开发人员可借助 DOM 提供的 API，轻松地删除、添加、替换或修改任何节点。
BOM （浏览器对象模型）
支持可以访问和操作浏览器窗口的浏览器对象模型，开发人员可以控制浏览器显示的页面以外的部分。

99、事件委托以及冒泡原理。

事件委托是利用冒泡阶段的运行机制来实现的，就是把一个元素响应事件的函数委托到另一个元素，一般是把一组元素的事件委托到他的父元素上，委托的优点是
减少内存消耗，节约效率
动态绑定事件
事件冒泡，就是元素自身的事件被触发后，如果父元素有相同的事件，如 onclick 事件，那么元素本身的触发状态就会传递，也就是冒到父元素，父元素的相同事件也会一级一级根据嵌套关系向外触发，直到 document/window，冒泡过程结束。

100、写个函数，可以转化下划线命名到驼峰命名

```
public static String UnderlineToHump(String para){
StringBuilder result=new StringBuilder();
String a[]=para.split("_");
for(String s:a){
if(result.length()==0){
result.append(s.toLowerCase());
}else{
result.append(s.substring(0, 1).toUpperCase());
result.append(s.substring(1).toLowerCase());
}
}
return result.toString();
}
}
```

101、深浅拷贝的区别和实现

数组的浅拷贝：
如果是数组，我们可以利用数组的一些方法，比如 slice，concat 方法返回一个新数组的特性来实现拷贝，但假如数组嵌套了对象或者数组的话，使用 concat 方法克隆并不完整，如果数组元素是基本类型，就会拷贝一份，互不影响，而如果是对象或数组，就会只拷贝对象和数组的引用，这样我们无论在新旧数组进行了修改，两者都会发生变化，我们把这种复制引用的拷贝方法称为浅拷贝，
深拷贝就是指完全的拷贝一个对象，即使嵌套了对象，两者也互相分离，修改一个对象的属性，不会影响另一个
如何深拷贝一个数组
1、这里介绍一个技巧，不仅适用于数组还适用于对象！那就是：

```
var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]
var new_arr = JSON.parse( JSON.stringify(arr) );
console.log(new_arr);
```

原理是 JOSN 对象中的 stringify 可以把一个 js 对象序列化为一个 JSON 字符串，parse 可以把 JSON 字符串反序列化为一个 js 对象，通过这两个方法，也可以实现对象的深复制。
但是这个方法不能够拷贝函数
浅拷贝的实现：
以上三个方法 concat,slice ,JSON.stringify 都是技巧类，根据实际项目情况

选择使用，我们可以思考下如何实现一个对象或数组的浅拷贝，遍历对象，然后把属性和属性值都放在一个新的对象里即可

```
var shallowCopy = function(obj) {
// 只拷贝对象
if (typeof obj !== 'object') return;
// 根据 obj 的类型判断是新建一个数组还是对象
```

复制代码

```
var newObj = obj instanceof Array ? [] : {};
// 遍历 obj，并且判断是 obj 的属性才拷贝
for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = obj[key];
}
}
return newObj;
}
```

深拷贝的实现

那如何实现一个深拷贝呢？说起来也好简单,我们在拷贝的时候判断一下属性值的类型，如果是对象，我们递归调用深拷贝函数不就好了~

```
var deepCopy = function(obj) {
if (typeof obj !== 'object') return;
var newObj = obj instanceof Array ? [] : {};
for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) : obj[key];
}
}
return newObj;
}
```

102、JS 中 string 的 startwith 和 indexof 两种方法的区别

JS 中 startwith 函数，其参数有 3 个，stringObj,要搜索的字符串对象，str，搜索的字符串，position，可选，从哪个位置开始搜索，如果以 position 开始的字符串以搜索字符串开头，则返回 true，否则返回 false

Indexof 函数,indexof 函数可返回某个指定字符串在字符串中首次出现的位置。

103、js 字符串转数字的方法

通过函数 parseInt（），可解析一个字符串，并返回一个整数，语法为 parseInt（string ,radix）
string：被解析的字符串
radix：表示要解析的数字的基数，默认是十进制，如果 radix<2 或>36,则返回 NaN

104、let const var 的区别 ，什么是块级作用域，如何用 ES5 的方法实现块级作用域（立即执行函数），ES6 呢

提起这三个最明显的区别是 var 声明的变量是全局或者整个函数块的，而 let,const 声明的变量是块级的变量，var 声明的变量存在变量提升，let,const 不存在，let 声明的变量允许重新赋值，const 不允许。

105、ES6 箭头函数的特性

ES6 增加了箭头函数，基本语法为
```
let func = value => value;
```
相当于
```
let func = function (value) {

return value;

};
```
箭头函数与普通函数的区别在于：
1、箭头函数没有 this，所以需要通过查找作用域链来确定 this 的值，这就意味着如果箭头函数被非箭头函数包含，this 绑定的就是最近一层非箭头函数的 this，
2、箭头函数没有自己的 arguments 对象，但是可以访问外围函数的 arguments 对象
3、不能通过 new 关键字调用，同样也没有 new.target 值和原型

106、setTimeout 和 Promise 的执行顺序

首先我们来看这样一道题：

```
setTimeout(function() {
console.log(1)
}, 0);
new Promise(function(resolve, reject) {
console.log(2)
for (var i = 0; i < 10000; i++) {
if(i === 10) {console.log(10)}
i == 9999 && resolve();
}
console.log(3)
}).then(function() {
console.log(4)
})
console.log(5);
```

输出答案为 2 10 3 5 4 1

要先弄清楚 settimeout（fun,0）何时执行，promise 何时执行，then 何时执行 settimeout 这种异步操作的回调，只有主线程中没有执行任何同步代码的前提下，才会执行异步回调，而 settimeout（fun,0）表示立刻执行，也就是用来改变任务的执行顺序，要求浏览器尽可能快的进行回调

promise 何时执行，由上图可知 promise 新建后立即执行，所以 promise 构造函数里代码同步执行的，

then 方法指向的回调将在当前脚本所有同步任务执行完成后执行，

那么 then 为什么比 settimeout 执行的早呢，因为 settimeout（fun,0）不是真的立即执行，

经过测试得出结论：执行顺序为：同步执行的代码-》promise.then->settimeout

107、有了解过事件模型吗，DOM0 级和 DOM2 级有什么区别，DOM 的分级是什么

JSDOM 事件流存在如下三个阶段：

事件捕获阶段

处于目标阶段

事件冒泡阶段

JSDOM 标准事件流的触发的先后顺序为：先捕获再冒泡，点击 DOM 节点时，事件传播顺序：事件捕获阶段，从上往下传播，然后到达事件目标节点，最后是冒泡阶段，从下往上传播

DOM 节点添加事件监听方法 addEventListener，中参数 capture 可以指定该监听是添加在事件捕获阶段还是事件冒泡阶段，为 false 是事件冒泡，为 true 是事件捕获，并非所有的事件都支持冒泡，比如 focus，blur 等等，我们可以通过 event.bubbles 来判断
事件模型有三个常用方法：
event.stopPropagation:阻止捕获和冒泡阶段中，当前事件的进一步传播，
event.stopImmediatePropagetion，阻止调用相同事件的其他侦听器，

event.preventDefault，取消该事件（假如事件是可取消的）而不停止事件的进一步传播，
event.target：指向触发事件的元素，在事件冒泡过程中这个值不变
event.currentTarget = this，时间帮顶的当前元素，只有被点击时目标元素的 target 才会等于 currentTarget，
最后，对于执行顺序的问题，如果 DOM 节点同时绑定了两个事件监听函数，一个用于捕获，一个用于冒泡，那么两个事件的执行顺序真的是先捕获在冒泡吗，答案是否定的，绑定在被点击元素的事件是按照代码添加顺序执行的，其他函数是先捕获再冒泡。

108、平时是怎么调试 JS 的

一般用 Chrome 自带的控制台

108、JS 的基本数据类型有哪些，基本数据类型和引用数据类型的区别，NaN 是什么的缩写，JS 的作用域类型，undefined==null 返回的结果是什么，undefined 与 null 的区别在哪，写一个函数判断变量类型

JS 的基本数据类型有字符串，数字，布尔，数组，对象，Null，Undefined,基本数据类型是按值访问的，也就是说我们可以操作保存在变量中的实际的值，
基本数据类型和引用数据类型的区别如下：

基本数据类型的值是不可变的，任何方法都无法改变一个基本类型的值，当这个变量重新赋值后看起来变量的值是改变了，但是这里变量名只是指向变量的一个指针，所以改变的是指针的指向改变，该变量是不变的，但是引用类型可以改变

基本数据类型不可以添加属性和方法，但是引用类型可以

基本数据类型的赋值是简单赋值，如果从一个变量向另一个变量赋值基本类型的值，会在变量对象上创建一个新值，然后把该值复制到为新变量分配的位置上，引用数据类型的赋值是对象引用，

基本数据类型的比较是值的比较，引用类型的比较是引用的比较，比较对象的内存地址是否相同

基本数据类型是存放在栈区的，引用数据类型同事保存在栈区和堆区

NaN 是 JS 中的特殊值，表示非数字，NaN 不是数字，但是他的数据类型是数字，它不等于任何值，包括自身，在布尔运算时被当做 false，NaN 与任何数运算得到的结果都是 NaN，党员算失败或者运算无法返回正确的数值的就会返回 NaN，一些数学函数的运算结果也会出现 NaN ，

JS 的作用域类型：

一般认为的作用域是词法作用域，此外 JS 还提供了一些动态改变作用域的方法，常见的作用域类型有：

函数作用域，如果在函数内部我们给未定义的一个变量赋值，这个变量会转变成为一个全局变量，

块作用域：块作用域吧标识符限制在{}中，

改变函数作用域的方法：

eval（），这个方法接受一个字符串作为参数，并将其中的内容视为好像在书写时就存在于程序中这个位置的代码，

with 关键字：通常被当做重复引用同一个对象的多个属性的快捷方式

undefined 与 null：目前 null 和 undefined 基本是同义的，只有一些细微的差别，null 表示没有对象，undefined 表示缺少值，就是此处应该有一个值但是还没有定义，因此 undefined==null 返回 false

此外了解== 和===的区别：

在做==比较时。不同类型的数据会先转换成一致后在做比较，===中如果类型不一致就直接返回 false，一致的才会比较

类型判断函数，使用 typeof 即可，首先判断是否为 null，之后用 typeof 哦按段，如果是 object 的话，再用 array.isarray 判断是否为数组，如果是数字的话用 isNaN 判断是否是 NaN 即可
扩展学习：

JS 采用的是词法作用域，也就是静态作用域，所以函数的作用域在函数定义的时候就决定了，

看如下例子：

```
var value = 1;
function foo() {
console.log(value);
}
function bar() {
var value = 2;
foo();
}
bar();
```

假设 JavaScript 采用静态作用域，让我们分析下执行过程：

执行 foo 函数，先从 foo 函数内部查找是否有局部变量 value，如果没有，就根据书写的位置，查找上面一层的代码，也就是 value 等于 1，所以结果会打印 1。

假设 JavaScript 采用动态作用域，让我们分析下执行过程：

执行 foo 函数，依然是从 foo 函数内部查找是否有局部变量 value。如果没有，就从调用函数的作用域，也就是 bar 函数内部查找 value 变量，所以结果会打印 2。

前面我们已经说了，JavaScript 采用的是静态作用域，所以这个例子的结果是 1。

109、setTimeout(fn,100);100 毫秒是如何权衡的

setTimeout()函数只是将事件插入了任务列表，必须等到当前代码执行完，主线程才会去执行它指定的回调函数，有可能要等很久，所以没有办法保证回调函数一定会在 setTimeout 指定的时间内执行，100 毫秒是插入队列的时间+等待的时间。

110、JS 的垃圾回收机制

GC（garbage collection），GC 执行时，中断代码，停止其他操作，遍历所有对象，对于不可访问的对象进行回收，在 V8 引擎中使用两种优化方法，
分代回收，2、增量 GC，目的是通过对象的使用频率，存在时长来区分新生代和老生代对象，多回收新生代区，少回收老生代区，减少每次遍历的时间，从而减少 GC 的耗时

回收方法：
引用计次，当对象被引用的次数为零时进行回收，但是循环引用时，两个对象都至少被引用了一次，因此导致内存泄漏，
标记清除

111、写一个 newBind 函数，完成 bind 的功能。

bind（）方法，创建一个新函数，当这个新函数被调用时，bind（）的第一个参数将作为它运行时的 this，之后的一序列参数将会在传递的实参前传入作为它的参数

```
Function.prototype.bind2 = function (context) {
if (typeof this !== "function") {
throw new Error("Function.prototype.bind - what is trying to be bound is not callable");
}
var self = this;
var args = Array.prototype.slice.call(arguments, 1);
var fNOP = function () {};
var fbound = function () {
self.apply(this instanceof self ? this : context, args.concat(Array.prototype.slice.call(arguments)));
}
fNOP.prototype = this.prototype;
fbound.prototype = new fNOP();
return fbound;
}
```

112、怎么获得对象上的属性：比如说通过 Object.key（）

从 ES5 开始，有三种方法可以列出对象的属性
for（let I in obj）该方法依次访问一个对象及其原型链中所有可枚举的类型
object.keys:返回一个数组，包括所有可枚举的属性名称
object.getOwnPropertyNames:返回一个数组包含不可枚举的属性

113、简单讲一讲 ES6 的一些新特性

ES6 在变量的声明和定义方面增加了 let、const 声明变量，有局部变量的概念，赋值中有比较吸引人的结构赋值，同时 ES6 对字符串、 数组、正则、对象、函数等拓展了一些方法，如字符串方面的模板字符串、函数方面的默认参数、对象方面属性的简洁表达方式，ES6 也 引入了新的数据类型 symbol，新的数据结构 set 和 map,symbol 可以通过 typeof 检测出来，为解决异步回调问题，引入了 promise 和 generator，还有最为吸引人了实现 Class 和模块，通过 Class 可以更好的面向对象编程，使用模块加载方便模块化编程，当然考虑到 浏览器兼容性，我们在实际开发中需要使用 babel 进行编译
重要的特性：
块级作用域：ES5 只有全局作用域和函数作用域，块级作用域的好处是不再需要立即执行的函数表达式，循环体中的闭包不再有问题
rest 参数：用于获取函数的多余参数，这样就不需要使用 arguments 对象了，
promise:一种异步编程的解决方案,比传统的解决方案回调函数和事件更合理强大
模块化：其模块功能主要有两个命令构成，export 和 import，export 命令用于规定模块的对外接口，import 命令用于输入其他模块提供的功能

114、call 和 apply 是用来做什么？

Call 和 apply 的作用是一模一样的，只是传参的形式有区别而已
1、改变 this 的指向
2、借用别的对象的方法，
3、调用函数，因为 apply，call 方法会使函数立即执行

115、了解事件代理吗，这样做有什么好处

事件代理/事件委托：利用了事件冒泡，只指定一个事件处理程序，就可以管理某一类型的事件，
简而言之：事件代理就是说我们将事件添加到本来要添加的事件的父节点，将事件委托给父节点来触发处理函数,这通常会使用在大量的同级元素需要添加同一类事件的时候，比如一个动态的非常多的列表，需要为每个列表项都添加点击事件，这时就可以使用事件代理，通过判断 e.target.nodeName 来判断发生的具体元素，这样做的好处是减少事件绑定，同事动态的 DOM 结构任然可以监听，事件代理发生在冒泡阶段

116、如何写一个继承？

原型链继承
核心： 将父类的实例作为子类的原型
特点：
非常纯粹的继承关系，实例是子类的实例，也是父类的实例
父类新增原型方法/原型属性，子类都能访问到
简单，易于实现
缺点：
要想为子类新增属性和方法，不能放到构造器中
无法实现多继承
来自原型对象的所有属性被所有实例共享
创建子类实例时，无法向父类构造函数传参
构造继承
核心：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）
特点：
解决了子类实例共享父类引用属性的问题
创建子类实例时，可以向父类传递参数
可以实现多继承（call 多个父类对象）
缺点：
实例并不是父类的实例，只是子类的实例
只能继承父类的实例属性和方法，不能继承原型属性/方法
无法实现函数复用，每个子类都有父类实例函数的副本，影响性能
实例继承
核心：为父类实例添加新特性，作为子类实例返回
特点：
不限制调用方式，不管是 new 子类()还是子类(),返回的对象具有相同的效果
缺点：
实例是父类的实例，不是子类的实例

不支持多继承
拷贝继承
特点：
支持多继承
缺点：
效率较低，内存占用高（因为要拷贝父类的属性）
组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
特点：
可以继承实例属性/方法，也可以继承原型属性/方法
既是子类的实例，也是父类的实例
不存在引用属性共享问题
可传参
函数可复用

寄生组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

117、给出以下代码，输出的结果是什么？原因？

```
for(var                                                    i=0;i<5;i++)
{ setTimeout(function(){ console.log(i); },1000); } console.log(i)
```

在一秒后输出 5 个 5
每次 for 循环的时候 setTimeout 都会执行，但是里面的 function 则不会执行被放入任务队列，因此放了 5 次；for 循环的 5 次执行完之后不到 1000 毫秒；1000 毫秒后全部执行任务队列中的函数，所以就是输出 5 个 5。

118、给两个构造函数 A 和 B，如何实现 A 继承 B？

```
function A(...) {}  A.prototype...
function B(...) {}  B.prototype...
A. prototype = Object.create(B.prototype);
// 再在 A 的构造函数里 new B(props);
for(var i = 0; i < lis.length; i++) {
lis[i].addEventListener('click', function(e) {
alert(i);
}, false)
}
```

119、能不能正常打印索引

在 click 的时候，已经变成 length 了

120、如果已经有三个 promise，A、B 和 C，想串行执行，该怎么写？

```
// promise
A.then(B).then(C).catch(...)
// async/await
(async ()=>{
await a();
await b();
await c();
})()
```

121、知道 private 和 public 吗

public：public 表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用
private：private 表示私有，私有的意思就是除了 class 自己之外，任何人都不可以直接使用

122、async 和 await 具体该怎么用？

```
(async () = > {
await new promise();
})()
```

123、知道哪些 ES6，ES7 的语法

promise，await/async，let、const、块级作用域、箭头函数

124、promise 和 await/async 的关系

都是异步编程的解决方案

125、js 的数据类型

字符串，数字，布尔，数组，null，Undefined，symbol，对象。

126、js 加载过程阻塞，解决方法。

指定 script 标签的 async 属性。
如果 async="async"，脚本相对于页面的其余部分异步地执行（当页面继续进行解析时，脚本将被执行）
如果不使用 async 且 defer="defer"：脚本将在页面完成解析时执行

127、js 对象类型，基本对象类型以及引用对象类型的区别

分为基本对象类型和引用对象类型
基本数据类型：按值访问，可操作保存在变量中的实际的值。基本类型值指的是简单的数据段。基本数据类型有这六种:undefined、null、string、number、boolean、symbol。
引用类型：当复制保存着对象的某个变量时，操作的是对象的引用，但在为对象添加属性时,操作的是实际的对象。引用类型值指那些可能为多个值构成的对象。引用类型有这几种：Object、Array、RegExp、Date、Function、特殊的基本包装类型(String、Number、Boolean)以及单体内置对象(Global、Math)。

128、JavaScript 中的轮播实现原理？假如一个页面上有两个轮播，你会怎么实现？

图片轮播的原理就是图片排成一行，然后准备一个只有一张图片大小的容器，对这个容器设置超出部分隐藏，在控制定时器来让这些图片整体左移或右移，这样呈现出来的效果就是图片在轮播了。
如果有两个轮播，可封装一个轮播组件，供两处调用

129、怎么实现一个计算一年中有多少周？

首先你得知道是不是闰年，也就是一年是 365 还是 366.
其次你得知道当年 1 月 1 号是周几。假如是周五，一年 365 天把 1 号 2 号 3 号减去，也就是把第一个不到一周的天数减去等于 362
还得知道最后一天是周几，加入是周五，需要把周一到周五减去，也就是 362-5=357.正常情况 357 这个数计算出来是 7 的倍数。357/7=51 。即为周数。

120、JS 的数据类型

字符串，数字，布尔，数组，null，Undefined，symbol，对象。

131、引用类型常见的对象

Object、Array、RegExp、Date、Function、特殊的基本包装类型(String、Number、Boolean)以及单体内置对象(Global、Math)等

132、es6 的常用

promise，await/async，let、const、块级作用域、箭头函数

133、class

ES6 提供了更接近传统语言的写法，引入了 Class（类）这个概念，作为对象的模板。通过 class 关键字，可以定义类。

134、口述数组去重

法一：indexOf 循环去重

法二：ES6 Set 去重；Array.from(new Set(array))
法三：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

135、call 和 apply 的区别

apply:调用一个对象的一个方法,用另一个对象替换当前对象。例如:B.apply(A, arguments);即 A 对象应用 B 对象的方法。
call：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.call(A, args1,args2);即 A 对象调用 B 对象的方法。

136、 es6 的常用特性

promise，await/async，let、const、块级作用域、箭头函数

137、箭头函数和 function 有什么区别

箭头函数根本就没有绑定自己的 this，在箭头函数中调用 this 时，仅仅是简单的沿着作用域链向上寻找，找到最近的一个 this 拿来使用

138、new 操作符原理

1. 创建一个类的实例：创建一个空对象 obj，然后把这个空对象的__proto__设置为构造函数的 prototype。
2. 初始化实例：构造函数被传入参数并调用，关键字 this 被设定指向该实例 obj。
3. 返回实例 obj。

139、bind,apply,call 是什么？

apply:调用一个对象的一个方法,用另一个对象替换当前对象。例如:B.apply(A, arguments);即 A 对象应用 B 对象的方法。
call:调用一个对象的一个方法,用另一个对象替换当前对象。例如:B.call(A, args1,args2);即 A 对象调用 B 对象的方法。
bind 除了返回是函数以外,它的参数和 call 一样。

140、bind 和 apply 的区别

返回不同:bind 返回是函数
参数不同:apply(A, arguments), bind(A, args1,args2)

141、promise 实现

```
function Promise(fn) {
var state = 'pending',
value = null,
callbacks = [];
this.then = function (onFulfilled, onRejected) {
return new Promise(function (resolve, reject) {
handle({
onFulfilled: onFulfilled || null,
onRejected: onRejected || null,
resolve: resolve,
reject: reject
});
});
};
function handle(callback) {
if (state === 'pending') {
callbacks.push(callback);
return;
}
var cb = state === 'fulfilled' ? callback.onFulfilled :
callback.onRejected,
ret;
if (cb === null) {
cb = state === 'fulfilled' ? callback.resolve : callback.reject;
cb(value);
return;
```

```
}
ret = cb(value);
callback.resolve(ret);
}
function resolve(newValue) {
if (newValue && (typeof newValue === 'object' || typeof newValue ===
'function')) {
var then = newValue.then;
if (typeof then === 'function') {
then.call(newValue, resolve, reject);
return;
}
}
state = 'fulfilled';
value = newValue;
execute();
}
function reject(reason) {
state = 'rejected';
value = reason;
execute();
}
function execute() {
setTimeout(function () {
callbacks.forEach(function (callback) {
handle(callback);
});
}, 0);
}
fn(resolve, reject);
}
```

149、assign 的深拷贝

```
function clone( obj ) {
var copy;
switch( typeof obj ) {
case "undefined":
break;
case "number":
copy = obj - 0;
break;
case "string":
copy = obj + "";
```

```
break;
case "boolean":
copy = obj;
break;
case "object":  //object 分为两种情况 对象（Object）和数组（Array）
if(obj === null) {
copy = null;
} else {
if( Object.prototype.toString.call(obj).slice(8, -1) === "Array") {
copy = [];
for( var i = 0 ; i < obj.length ; i++ ) {
copy.push(clone(obj[i]));
}
} else {
copy = {};
for( var j in obj) {
copy[j] = clone(obj[j]);
}
}
}
break;
default:
copy = obj;
break;
}
return copy;
}
```

142、没有 promise 怎么办

没有 promise，可以用回调函数代替

143、事件委托

把一个元素响应事件（click、keydown.....）的函数委托到另一个元素；
优点：减少内存消耗、动态绑定事件。

144、箭头函数和 function 的区别

箭头函数根本就没有绑定自己的 this，在箭头函数中调用 this 时，仅仅是简单的沿着作用域链向上寻找，找到最近的一个 this 拿来使用

145、arguments 是什么？

arguments 是类数组对象，有 length 属性，不能调用数组方法
可用 Array.from()转换

146、箭头函数获取 arguments

可用…rest 参数获取

147、事件代理是什么？

事件代理是利用事件的冒泡原理来实现的，何为事件冒泡呢？就是事件从最深的节点开始，然后逐步向上传播事件，举个例子：页面上有这么一个节点树，div>ul>li>a;比如给最里面的 a 加一个 click 点击事件，那么这个事件就会一层一层的往外执行，执行顺序 a>li>ul>div，有这样一个机制，那么我们给最外面的 div 加点击事件，那么里面的 ul，li，a 做点击事件的时候，都会冒泡到最外层的 div 上，所以都会触发，这就是事件代理，代理它们父级代为执行事件。

148、Eventloop

任务队列中，在每一次事件循环中，macrotask 只会提取一个执行，而 microtask 会一直提取，直到 microsoft 队列为空为止。
也就是说如果某个 microtask 任务被推入到执行中，那么当主线程任务执行完成后，会循环调用该队列任务中的下一个任务来执行，直到该任务队列到最后一个任务为止。而事件循环每次只会入栈一个 macrotask，主线程执行完成该任务后又会检查 microtasks 队列并完成里面的所有任务后再执行 macrotask 的任务。
macrotasks: setTimeout, setInterval, setImmediate, I/O, UI rendering

microtasks: process.nextTick, Promise, MutationObserver

149、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

150、介绍 JavaScript 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

151、 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

152、 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。

153、 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。

154、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。

155、JavaScript 实现继承的 3 种方法

1)、借用构造函数法（又叫经典继承）

```
function SuperType(name) {
    this.name = name;
    this.sayName = function() {        window.alert(this.name);    };}
function SubType(name, age) {
    SuperType.call(this, name); //在这里借用了父类的构造函数
    this.age = age;}
```

2)、对象冒充

```
function SuperType(name) {
    this.name = name;                    this.sayName = function() {        window.alert(this.name);    };}        function SubType(name, age) {
    this.supertype = SuperType; // 在这里使用了对象冒充 this.supertype(name);
    this.age = age;}
```

3)、组合继承（最常用）

```
function SuperType(name) {
    this.name = name;
}          SuperType.prototype = {                sayName : function() {        window.alert(this.name);    }};        function SubType(name, age) {        SuperType.call(this, name); //在这里继承属性    this.age = age;}
SubType.prototype = new SuperType(); //这里继承方法
```

156、JavaScript 定义类的 4 种方法

1)、工厂方法

```
function creatPerson(name, age) {                var obj = new Object();
    obj.name = name;    obj.age = age;
    obj.sayName = function() {         window.alert(this.name);    };
return obj;}
```

2)、构造函数方法

```
function Person(name, age) {
    this.name = name;    this.age = age;
    this.sayName = function() {        window.alert(this.name);    };}
```

3)、原型方法

```
function Person() {              }              Person.prototype = {    constructor : Person,    name : "Ning",    age : "23",    sayName :
```

function() { window.alert(this.name); }};
大家可以看到这种方法有缺陷，类里属性的值都是在原型里给定的。

4)、组合使用构造函数和原型方法（使用最广）
function Person(name, age) { this.name = name; this.age = age;}
Person.prototype = { constructor : Person, sayName : function()
{ window.alert(this.name); }};
将构造函数方法和原型方法结合使用是目前最常用的定义类的方法。这种方法的好处是实现了属性定义和方法定义的分离。比如我可以创建两个对象 person1 和 person2，它们分别传入各自的 name 值和 age 值，但 sayName()方法可以同时使用原型里定义的。

168、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。

157、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。

158、 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）。

159、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。

160、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

161、 ["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。

162、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;）

163、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。

164、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');}

165、 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 。

166、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。

167、 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

168、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。

169、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。

170、什么是跨域？

要明白什么是跨域之前，首先要明白什么是同源策略？
同源策略就是用来限制从一个源加载的文档或脚本与来自另一个源的资源进行交互。那怎样判断是否是同源呢？
如果协议，端口（如果指定了）和主机对于两个页面是相同的，则两个页面具有相同的源，也就是同源。也就是说，要同时满足以下 3 个条件，才能叫同源：
协议相同
端口相同
主机相同
举个例子就一目了然了：
我们来看下面的页面是否与 http://store.company.com/dir/index.html 是同源的？
http://store.company.com/dir/index2.html 同源
http://store.company.com/dir2/index3.html 同源 虽然在不同文件夹下
https://store.company.com/secure.html 不同源 不同的协议(https)
http://store.company.com:81/dir/index.html 不同源 不同的端口(81)
http://news.company.com/dir/other.html 不同源 不同的主机(news)

171、跨域的几种解决方案

1)、document.domain 方法
我们来看一个具体场景：有一个页面 http://www.example.com/a.html ，它里面有一个 iframe，这个 iframe 的源是 http://example.com/b.html ，很显然它们是不同源的，所以我们无法在父页面中操控子页面的内容。
解决方案如下：
<!-- b.html --><script>document.domain = 'example.com';</script>
<!-- a.html --><script>document.domain = 'example.com';var iframe = document.getElementById('iframe').contentWindow.document;
//后面就可以操作 iframe 里的内容了...</script>
所以我们只要将两个页面的 document.domain 设置成一致就可以了，要注意的是，document.domain 的设置是有限制的，我们只能把 document.domain 设置成自身或更高一级的父域。
但是，这种方法只能解决主域相同的跨域问题。
2)、window.name 方法
window 对象有个 name 属性，该属性有个特征：即在一个窗口(window)的生命周期内，窗口载入的所有的页面都是共享一个 window.name 的，每个页面对 window.name 都有读写的权限，window.name 是持久存在一个窗口载入过的所有页面中的，并不会因新页面的载入而进行重置。
我们来看一个具体场景，在一个页面 example.com/a.html 中，我们想获取 data.com/data.html 中的数据，以下是解决方案：
<!-- data.html --><script>window.name = 'data'; //这是就是我们需要通信的数据</script>
<!-- a.html --><html><head><script> function getData () { var

iframe = document.getElementById('iframe'); iframe.src = 'example.com/b.html'; // 这里让 iframe 与父页面同源 iframe.onload = function () { var data = iframe.contentWindow.name; //在这里我们得到了跨域页面中传来的数据 }; }</script></head><body></body></html>

3)、JSONP 方法

JONSP(JSON with Padding)是 JSON 的一种使用模式。基本原理如下：

<!-- a.html --><script> function dealData (data) { console.log(data); }</script>

<script src='http://example.com/data.php?callback=dealData'></script>

<?php $callback = $_GET['callback']; $data = 'data'; echo $callback.'('.json_encode($data).')';?>

这时候在 a.html 中我们得到了一条 js 的执行语句 dealData('data')，从而达到了跨域的目的。

所以 JSONP 的原理其实就是利用引入 script 不限制源的特点，把处理函数名作为参数传入，然后返回执行语句，仔细阅读以上代码就可以明白里面的意思了。

如果在 jQuery 中用 JSONP 的话就更加简单了：

<script>$.getJSON(''http://example.com/data.php?callback=?', function (data) { console.log(data);});</script>

注意 jQuery 会自动生成一个全局函数来替换 callback=?中的问号，之后获取到数据后又会自动销毁，实际上就是起一个临时代理函数的作用。$.getJSON 方法会自动判断是否跨域，不跨域的话，就调用普通的 ajax 方法；跨域的话，则会以异步加载 js 文件的形式来调用 JSONP 的回调函数。

172、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 中嵌入了一个 http://www.xxx.com/test.js

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" charset="utf-8"></script>

173、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。

174、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。

```
// CMD
define(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();
})
```

175、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？）

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。

176、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。

177、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

178、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

179、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

180、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。

181、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

182、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。

183、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。

184、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。

185、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。

186、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

187、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。

188、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()

189、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）

190、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)

191、 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。

前端面试题集锦——浏览器

1. 跨标签页通讯

不同标签页间的通讯，本质原理就是去运用一些可以 共享的中间介质，因此比较常用的有以下方法:
通过父页面 window.open()和子页面 postMessage
o 异步下，通过 window.open('about: blank') 和 tab.location.href = '*'
设置同域下共享的 localStorage 与监听 window.onstorage
o 重复写入相同的值无法触发
o 会受到浏览器隐身模式等的限制
设置共享 cookie 与不断轮询脏检查(setInterval)
借助服务端或者中间层实现

2. 浏览器架构

用户界面
主进程
内核
o 渲染引擎
oJS 引擎
执行栈
o 事件触发线程
消息队列
微任务
宏任务
o 网络异步线程
o 定时器线程

3. 浏览器下事件循环(Event Loop)

事件循环是指: 执行一个宏任务，然后执行清空微任务列表，循环再执行宏任务，再清微任务列表
微任务 microtask(jobs): promise / ajax / Object.observe(该方法已废弃)
宏任务 macrotask(task): setTimout / script / IO / UI Rendering

4. 从输入 url 到展示的过程

DNS 解析
TCP 三次握手
发送请求，分析 url，设置请求报文(头，主体)
服务器返回请求的文件 (html)
浏览器渲染
oHTML parser --> DOM Tree
标记化算法，进行元素状态的标记
dom 树构建
oCSS parser --> Style Tree
解析 css 代码，生成样式树
oattachment --> Render Tree
结合 dom 树 与 style 树，生成渲染树
olayout: 布局
oGPU painting: 像素绘制页面

5. 重绘与回流

当元素的样式发生变化时，浏览器需要触发更新，重新绘制元素。这个过程中，有两种类型的操作，即重绘与回流。
重绘(repaint): 当元素样式的改变不影响布局时，浏览器将使用重绘对元素进行更新，此时由于只需要 UI 层面的重新像素绘制，因此 损耗较少
回流(reflow): 当元素的尺寸、结构或触发某些属性时，浏览器会重新渲染页面，称为回流。此时，浏览器需要重新经过计算，计算后还需要重新页面布局，因此是较重的操作。会触发回流的操作:
o 页面初次渲染
o 浏览器窗口大小改变
o 元素尺寸、位置、内容发生改变
o 元素字体大小变化
o 添加或者删除可见的 dom 元素
o 激活 CSS 伪类（例如：:hover）
o 查询某些属性或调用某些方法
clientWidth、clientHeight、clientTop、clientLeft
offsetWidth、offsetHeight、offsetTop、offsetLeft
scrollWidth、scrollHeight、scrollTop、scrollLeft
getComputedStyle()
getBoundingClientRect()
scrollTo()
回流必定触发重绘，重绘不一定触发回流。重绘的开销较小，回流的代价较高。

最佳实践:
css
o 避免使用 table 布局
o 将动画效果应用到 position 属性为 absolute 或 fixed 的元素上
javascript
o 避免频繁操作样式，可汇总后统一 一次修改
o 尽量使用 class 进行样式修改
o 减少 dom 的增删次数，可使用 字符串 或者 documentFragment 一次性插入
o 极限优化时，修改样式可将其 display: none 后修改
o 避免多次触发上面提到的那些会触发回流的方法，可以的话尽量用 变量存住

6. 存储

我们经常需要对业务中的一些数据进行存储，通常可以分为 短暂性存储 和 持久性储存。
短暂性的时候，我们只需要将数据存在内存中，只在运行时可用
持久性存储，可以分为 浏览器端 与 服务器端
o 浏览器:
  cookie: 通常用于存储用户身份，登录状态等
  http 中自动携带， 体积上限为 4K， 可自行设置过期时间
  localStorage / sessionStorage: 长久储存/窗口关闭删除， 体积限制为 4~5M
  indexDB
o 服务器:
  分布式缓存 redis
  数据库

7. Web Worker

现代浏览器为 JavaScript 创造的 多线程环境。可以新建并将部分任务分配到 worker 线程并行运行，两个线程可 独立运行，互不干扰，可通过自带的 消息机制 相互通信。
基本用法:
```
// 创建 workerconst worker = new Worker('work.js');
// 向 worker 线程推送消息
worker.postMessage('Hello World');
// 监听 worker 线程发送过来的消息
worker.onmessage = function (event) {
  console.log('Received message ' + event.data);
}
```
限制:
  同源限制

无法使用 document / window / alert / confirm
无法加载本地资源

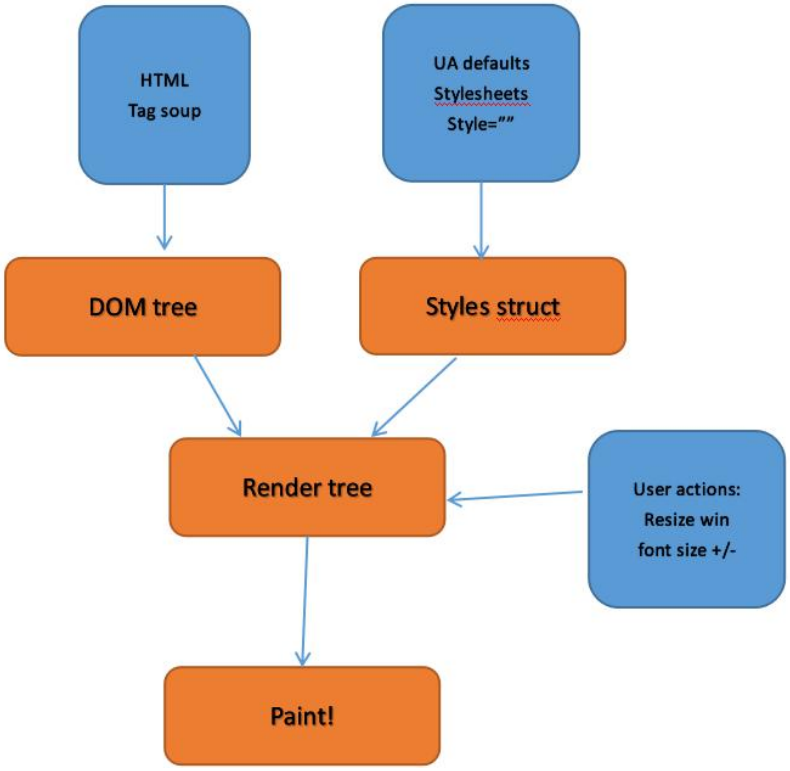
8. V8 垃圾回收机制

垃圾回收: 将内存中不再使用的数据进行清理，释放出内存空间。V8 将内存分成 新生代空间 和 老生代空间。
新生代空间: 用于存活较短的对象
o 又分成两个空间: from 空间 与 to 空间
oScavenge GC 算法: 当 from 空间被占满时，启动 GC 算法
存活的对象从 from space 转移到 to space
清空 from space
from space 与 to space 互换
完成一次新生代 GC
老生代空间: 用于存活时间较长的对象
o 从 新生代空间 转移到 老生代空间 的条件
经历过一次以上 Scavenge GC 的对象
当 to space 体积超过 25%
o 标记清除算法: 标记存活的对象，未被标记的则被释放
增量标记: 小模块标记，在代码执行间隙执，GC 会影响性能
并发标记(最新技术): 不阻塞 js 执行
o 压缩算法: 将内存中清除后导致的碎片化对象往内存堆的一端移动，解决 内存的碎片化

9. 内存泄露

意外的全局变量: 无法被回收
定时器: 未被正确关闭，导致所引用的外部变量无法被释放
事件监听: 没有正确销毁 (低版本浏览器可能出现)
闭包: 会导致父级中的变量无法被释放
dom 引用: dom 元素被删除时，内存中的引用未被正确清空
可用 chrome 中的 timeline 进行内存标记，可视化查看内存的变化情况，找出异常点。

10. reflow(回流)和 repaint(重绘)优化



浏览器渲染过程: DOM tree, CSS tree --> Render tree --> Paint
DOM tree 根节点为 html
渲染从浏览器左上角到右下角
第一次打开页面至少触发一次重绘和回流，结构如宽高位置变化时，触发 reflow 回流;非结构如背景色变化时，触发 repaint 重绘. 二者都会造成体验不佳

11.如何减少重绘和回流?

通过 classname 或 cssText 一次性修改样式，而非一个一个改
离线模式: 克隆要操作的结点，操作后再与原始结点交换，类似于虚拟 DOM
避免频繁直接访问计算后的样式，而是先将信息保存下来
绝对布局的 DOM，不会造成大量 reflow
p 不要嵌套太深，不要超过六层。

12.一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么?

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 Tree）；
载入解析到的资源文件，渲染页面，完成。

13.localStorage 与 sessionStorage 与 cookie 的区别总结

共同点：都保存在浏览器端，且同源
localStorage 与 sessionStorage 统称 webStorage,保存在浏览器,不参与服务器通信,大小为 5M
生命周期不同: localStorage 永久保存, sessionStorage 当前会话，都可手动清除
作用域不同：不同浏览器不共享 local 和 session，不同会话不共享 session
Cookie：设置的过期时间前一直有效，大小 4K.有个数限制，各浏览器不同，一般为 20 个.携带在 HTTP 头中，过多会有性能问题.可自己封装，也可用原生。

14.浏览器如何阻止事件传播，阻止默认行为

阻止事件传播(冒泡): e.stopPropagation()
阻止默认行为: e.preventDefault()

15.虚拟 DOM 方案相对原生 DOM 操作有什么优点，实现上是什么原理？

虚拟 DOM 可提升性能，无须整体重新渲染，而是局部刷新.
JS 对象, diff 算法。

16.浏览器事件机制中事件触发三个阶段

事件捕获阶段：从 dom 树节点往下找到目标节点，不会触发函数
事件目标处理函数：到达目标节点
事件冒泡：最后从目标节点往顶层元素传递，通常函数在此阶段执行.
addEventListener 第三个参数默认 false(冒泡阶段执行),true(捕获阶段执行).

阻止冒泡见以上方法。

17.什么是跨域？为什么浏览器要使用同源策略？你有几种方式可以解决跨域问题？了解预检请求嘛？

跨域是指一个域下的文档或脚本试图去请求另一个- 域下的资源
防止 XSS、CSFR 等攻击，协议+域名+端口不同
jsonp；跨域资源共享（CORS）(Access control)；服务器正向代理等

```
function jsonp({ url, params, callback }) { //接收参数配置
    return new Promise((resolve, reject) => { // es6 promise
        Let script = document.createElement('script');//创建 script 标签
        window[callback] = function (data) {
            resolve(data);
            document.body.removeChild(script); //移除 script 标签
        }
        document.body.removeChild(script); //移除 script 标签
        params = { ...params, callback };
        Let arr = [];
        for (let key in params) {
            arr.push(`${key} = !${params[key]}`);
        }
        script.src = ` ${url}?${arr.jion('&')}` //拼接 ur1 地址
        document.body.appendChild(script); //  将创建好的 sCr ipt 标签添加到 body 下面
    })
}
```

预检请求：需预检的请求要求必须首先使用 OPTIONS 方法发起一个预检请求到服务器，以获知服务器是否允许该实际请求。"预检请求“的使用，可以避免跨域请求对服务器的用户数据产生未预期的影响。

18.了解浏览器缓存机制吗？

浏览器缓存就是把一个已经请求过的资源拷贝一份存储起来，当下次需要该资源时，浏览器会根据缓存机制决定直接使用缓存资源还是再次向服务器发送请求.
from memory cache ; from disk cache
作用：减少网络传输的损耗以及降低服务器压力。
优先级：强制缓存 > 协商缓存; cache-control > Expires > Etag > Last-modified

19.为什么操作 DOM 慢?

DOM 本身是一个 js 对象，操作这个对象本身不慢，但是操作后触发了浏览器的行为，如 repaint 和 reflow 等浏览器行为，使其变慢。

20.什么情况会阻塞渲染?

js 脚本同步执行
css 和图片虽然是异步加载，但 js 文件执行需依赖 css，所以 css 也会阻塞渲染。

21.如何判断 js 运行在浏览器中还是 node 中?

判断有无全局对象 global 和 window

22.关于 web 以及浏览器处理预加载有哪些思考?

图片等静态资源在使用之前就提前请求
资源使用到的时候能从缓存中加载，提升用户体验
页面展示的依赖关系维护

23.http 多路复用

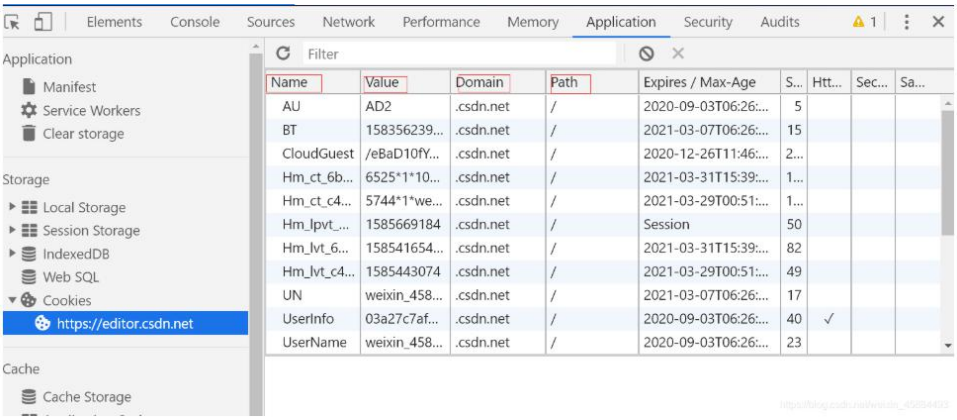
Keep-Alive: Keep-Alive 解决的核心问题：一定时间内，同一域名多次请求数据，只建立一次 HTTP 请求，其他请求可复用每一次建立的连接通道，以达到提高请求效率的问题。这里面所说的一定时间是可以配置的，不管你用的是 Apache 还是 nginx。
解决两个问题：串行文件传输(采用二进制数据帧)；连接数过多(采用流，并行传输)

24.http 和 https：

http: 最广泛网络协议，BS 模型，浏览器高效。
https: 安全版，通过 SSL 加密，加密传输，身份认证，密钥
1 https 相对于 http 加入了 ssl 层，加密传输，身份认证;
2 需要到 ca 申请收费的证书;
3 安全但是耗时多，缓存不是很好;
4 注意兼容 http 和 https;
5 连接方式不同，端口号也不同, http 是 80, https 是 443

25. cookie 可设置哪些属性？httponly?

chrome 控制台的 application 下可查看:



name 字段为一个 cookie 的名称。
value 字段为一个 cookie 的值。
domain 字段为可以访问此 cookie 的域名。
path 字段为可以访问此 cookie 的页面路径。比如 domain 是 abc.com,path 是/test，那么只有/test 路径下的页面可以读取此 cookie。
expires/Max-Age 字段为此 cookie 超时时间。若设置其值为一个时间，那么当到达此时间后，此 cookie 失效。不设置的话默认值是 Session，意思是 cookie 会和 session 一起失效。当浏览器关闭(不是浏览器标签页，而是整个浏览器) 后，此 cookie 失效。
Size 字段 此 cookie 大小。
http 字段 cookie 的 httponly 属性。若此属性为 true，则只有在 http 请求头中会带有此 cookie 的信息，而不能通过 document.cookie 来访问此 cookie。
secure 字段 设置是否只能通过 https 来传递此条 cookie

26.登录后，前端做了哪些工作，如何得知已登录

前端存放服务端下发的 cookie，简单说就是写一个字段在 cookie 中表明已登录，并设置失效日期

或使用后端返回的 token，每次 ajax 请求将 token 携带在请求头中，这也是防范 csrf 的手段之一。

27.http 状态码

1**：服务器收到请求，需请求者进一步操作
2**：请求成功
3**：重定向，资源被转移到其他 URL 了
4**：客户端错误，请求语法错误或没有找到相应资源
5**：服务端错误, server error
301：资源(网页等)被永久转移到其他 URL，返回值中包含新的 URL，浏览器会自动定向到新 URL
302：临时转移．客户端应访问原有 URL
304: Not Modified．指定日期后未修改，不返回资源
403：服务器拒绝执行请求
404：请求的资源(网页等)不存在
500：内部服务器错误

28.# Http 请求头缓存设置方法

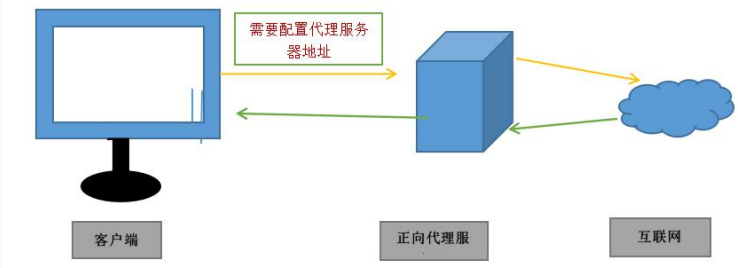
Cache-control, expire, last-modify

29.实现页面回退刷新

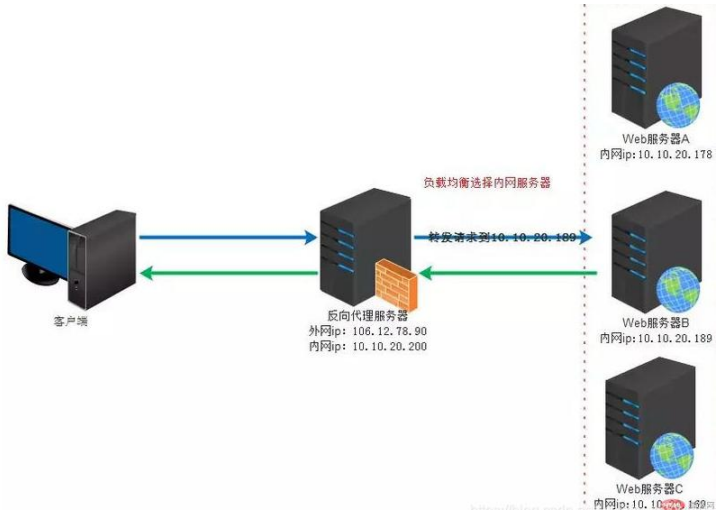
旧: window.history.back() + window.location.href=document.referrer;
新: HTML5 的新 API 扩展了 window.history，使历史记录点更加开放了。可以存储当前历史记录点、替换当前历史记录点、监听历史记录点 onpopstate, replaceState

30.　正向代理和反向代理

正向代理：



（1）访问原来无法访问的资源，如 google
（2） 可以做缓存，加速访问资源
（3）对客户端访问授权，上网进行认证
（4）代理可以记录用户访问记录（上网行为管理），对外隐藏用户信息
反向代理：



（1）保证内网的安全，可以使用反向代理提供 WAF 功能，阻止 web 攻击大型网站，通常将反向代理作为公网访问地址，Web 服务器是内网。
（2）负载均衡，通过反向代理服务器来优化网站的负载

31.关于预检请求

在非简单请求且跨域的情况下，浏览器会自动发起 options 预检请求。

32.三次握手四次挥手

开启连接用三次握手，关闭用四次挥手。

33.TCP 和 UDP 协议

TCP（Transmission Control Protocol：传输控制协议；面向连接，可靠传输
UDP（User Datagram Protocol）：用户数据报协议；面向无连接，不可靠传输

34.进程和线程的区别

进程：是并发执行的程序在执行过程中分配和管理资源的基本单位，是一个动态概念，竞争计算机系统资源的基本单位。
线程：是进程的一个执行单元，是进程内科调度实体。比进程更小的独立运行的基本单位。线程也被称为轻量级进程。
一个程序至少一个进程，一个进程至少一个线程。

35.浏览器内核

浏览器内核可以分成两部分： 渲染引擎(layout engineer 或者 Rendering Engine)和 JS 引擎。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核。

36.渲染引擎

渲染引擎负责取得网页的内容（HTML、XML、图像等等）、整理讯息（例如加入 CSS 等），以及计算网页的显示方式，然后会输出至显示器或打印机。

37.JS 引擎

JS 引擎则是解析 Javascript 语言，执行 javascript 语言来实现网页的动态效果。
最开始渲染引擎和 JS 引擎并没有区分的很明确，后来 JS 引擎越来越独立，内核就倾向于只指渲染引擎。

38.主流浏览器内核

常见的浏览器内核可以分为四种：Trident，Gecko，Webkit，Chromium/Bink。内核看着陌生，那么转换为相对应的浏览器：IE，Mozilla FireFox，Safari，Chrome 是不是立马就觉得熟悉了。

39.rident 内核常见浏览器

（1）IE6、IE7、IE8（Trident 4.0）、IE9（Trident 5.0）、IE10（Trident 6.0）；
（2）猎豹安全浏览器：1.0-4.2 版本为 Trident+Webkit，4.3 版本为 Trident+Blink；
（3）360 安全浏览器 ：1.0-5.0 为 Trident，6.0 为 Trident+Webkit，7.0 为 Trident+Blink；
（4）360 极速浏览器：7.5 之前为 Trident+Webkit,7.5 为 Trident+Blink；
（5）傲游浏览器 ：傲游 1.x、2.x 为 IE 内核，3.x 为 IE 与 Webkit 双核；
（6）搜狗高速浏览器：1.x 为 Trident，2.0 及以后版本为 Trident+Webkit；

40.开源内核

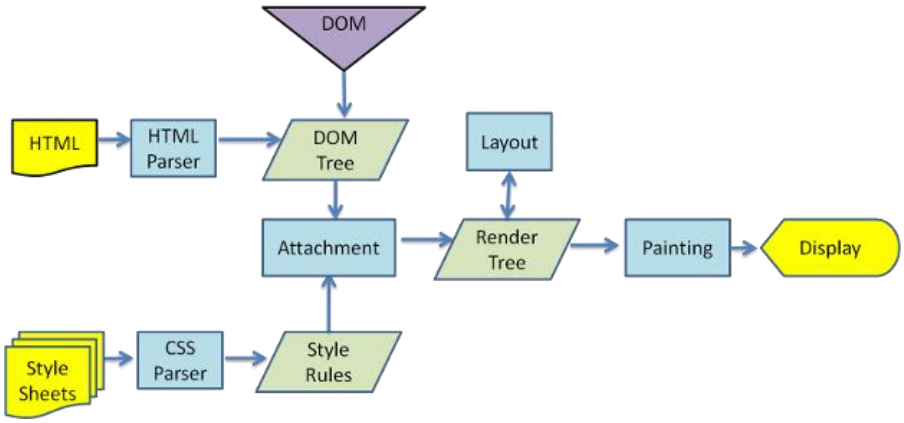
Gecko(Firefox 内核)：Netscape6 开始采用的内核，后来的 Mozilla FireFox(火狐浏览器) 也采用了该内核，Gecko 的特点是代码完全公开，因此，其可开发程度很高，全世界的程序员都可以为其编写代码，增加功能。因为这是个开源内核，因此受到许多人的青睐，Gecko 内核的浏览器也很多，这也是 Gecko 内核虽然年轻但市场占有率能够迅速提高的重要原因。

41.Firefox 内核

事实上，Gecko 引擎的由来跟 IE 不无关系，前面说过 IE 没有使用 W3C 的标准，这导致了微软内部一些开发人员的不满；他们与当时已经停止更新了的 Netscape 的一些员工一起创办了 Mozilla，以当时的 Mosaic 内核为基础重新编写内核，于是开发出了 Gecko。不过事实上，Gecko 内核的浏览器仍然还是 Firefox (火狐) 用户最多，所以有时也会被称为 Firefox 内核。此外 Gecko 也是一个跨平台内核，可以在 Windows、 BSD、Linux 和 Mac OS X 中使用。

42.描述浏览器渲染过程

DOM Tree 和 CSS RULE Tree 将 html 和 css 解析成树形数据结构，然后 Dom 和 css 合并后生成 Render Tree，用 layout 来确定节点的位置以及关系，通过 painting 按照规则来画到屏幕上，由 display 打击最终看到效果。



43.什么是 DOCTYPE 及作用？

DTD（Document Type Definition）：文档类型定义
作用：浏览器会使用 DTD 来判断文本类型

44.常见的 DOCTYPE 声明有几种？

1、**HTML 4.01 Strict**:（严格的）

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

2、**HTML 4.01 Transitional**:（传统的）

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

1、**HTML 5**:

<!DOCTYPE html>

45.什么是 Reflow？

Dom 节点中的元素都有自己的盒子模型，浏览器根据样式进行计算，将元素放在它该出现的位置这成为 Reflow

46.什么是 Repaint？

当盒子的位置大小属性都确定好了之后，浏览器便会按照特性来绘制最终显示在屏幕上，页面的内容出现，我们称之为 Repaint
每当我们增加修改删除 DOM 节点、移动 DOM 的位置、更改 CSS 样式的时候，就会触发 Reflow 和 Repaint，容易造成资源浪费，那我们应该如何如何减少 Reflow 和 Repaint 频率呢？
1.避免在 document 上直接进行频繁的 DOM 操作，可以将元素 display 设置为“none”或者如果需要创建多个 DOM 节点，可以使用 DocumentFragment 创建完，然后一次性地假如 document
2.集中修改样式，尽可能少的修改元素 style 上的样式
3.缓存 Layout 属性值，对于 Layout 属性中非引用类型的值（数字型），如果需要多次访问则可以在一次访问时先存储到局部变量中，之后都使用局部变量，这样可以避免每次读取属性时造成浏览器的渲染。
4.权衡速度的平滑：比如实现一个动画，以 1 个像素为单位移动这样最平滑，但 reflow 就会过于频繁，CPU 很快就会被完全占用。如果以 3 个像素为单位移动就会好很多。

47.从浏览器地址栏输入 url 到显示页面的步骤

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、图像等）；
浏览器对加载到的资源（HTML、JS、CSS 等）进行语法解析，建立相对应的内部数据结构（如 HTML 的 DOM）；
载入解析到的资源文件，渲染页面，完成。

48.请描述一下 cookie、sessionStorage、localStorage 的区别

存储大小：cookie 数据不能超过 4k，sessionStorage 和 localStorage，可以达到 5M 或更大
数据有效期：sessionStorage 关闭标签就去清除；localStorage 需要手动清除；cookie 在设置的 cookie 过期时间之前一直有效
请求数据，cookie 可以作为请求参数去请求服务器接口，即 cookie 在浏览器和服务器间来回传递
作用范围：cookie 只属于某个路径下，需要设置路径，同源窗口共享，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
如何实现浏览器内多个标签页之间的通信
localstorage、 cookies 等本地存储方式
使用 url 带参数做页面间的跳转
可以把数据传给后端，在另一个页面再去请求后端的接口拿数据

49.什么是 XXS 攻击

Cross-site script，跨站脚本攻击，
当其它用户浏览该网站时候，该段 HTML 代码会自动执行，从而达到攻击的目的，如盗取用户的 Cookie，破坏页面结构，重定向到其它网站等。
XSS 类型：
一般可以分为： 持久型 XSS 和非持久性 XSS
持久型 XSS 就是对客户端攻击的脚本植入到服务器上，从而导致每个正常访问到的用户都会遭到这段 XSS 脚本的攻击。(如上述的留言评论功能)
非持久型 XSS 是对一个页面的 URL 中的某个参数做文章，把精心构造好的恶意脚本包装在 URL 参数重，再将这个 URL 发布到网上，骗取用户访问，从而进行攻击。

50.CSRF 攻击

CSRF(Cross-site request forgery), 中文名称：跨站请求伪造
CSRF 可以简单理解为：攻击者盗用了你的身份，以你的名义发送恶意请求，容易造成个人隐私泄露以及财产安全。
防范：

post 请求
使用 token
验证码

51.DDOS 攻击

利用目标系统网络服务功能缺陷或者直接消耗其系统资源，使得该目标系统无法提供正常的服务。
DDoS 攻击通过大量合法的请求占用大量网络资源，以达到瘫痪网络的目的。 具体有几种形式：
通过使网络过载来干扰甚至阻断正常的网络通讯； 通过向服务器提交大量请求，使服务器超负荷； 阻断某一用户访问服务器； 阻断某服务与特定系统或个人的通讯。

52.从 URL 输入到页面展现到底发生什么

DNS 解析:将域名解析成 IP 地址
TCP 连接：TCP 三次握手
发送 HTTP 请求
服务器处理请求并返回 HTTP 报文
浏览器解析渲染页面
断开连接：TCP 四次挥手
TCP 三次握手结束后，开始发送 HTTP 请求报文。 请求报文由请求行（request line）、请求头（header）、请求体
1.请求行包含请求方法、URL、协议版本 2.请求头包含请求的附加信息，由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。 比如：Host，表示主机名，虚拟主机；Connection,HTTP/1.1 增加的，使用 keepalive，即持久连接，一个连接可以发多个请求；User-Agent，请求发出者，兼容性以及定制化需求。
3.请求体，可以承载多个请求参数的数据，包含回车符、换行符和请求数据，并不是所有请求都具有请求数据
首先浏览器发送过来的请求先经过控制器，控制器进行逻辑处理和请求分发，接着会调用模型，这一阶段模型会获取 redis db 以及 MySQL 的数据，获取数据后将渲染好的页面，响应信息会以响应报文的形式返回给客户端，最后浏览器通过渲染引擎将网页呈现在用户面前。
根据 HTML 解析出 DOM 树
根据 CSS 解析生成 CSS 规则树
结合 DOM 树和 CSS 规则树，生成渲染树
根据渲染树计算每一个节点的信息

根据计算好的信息绘制页面
当数据传送完毕，需要断开 tcp 连接，此时发起 tcp 四次挥手。

53.cookie 中存放的数据

在最初了解 cookie 的时候，曾经看到有介绍说在 cookie 中存储用户名和密码，方便用户之后登陆，然而，真的是可以的嘛？安全性问题怎么解决呢？
此外，在建立一次会话也就是 session 时，常用 cookie 来传递 SessionID。用户注册之后，服务器会设置 http 响应头 Set-Cookie 把 SessionID 随机数传递到浏览器。浏览器得到 cookie 之后，每次上传到服务器都要带上 cookie。服务器识别 cookie，可以得到用户的信息，用户不需要每次都登陆。（这也是每次 请求都要携带 cookie 的原因之一）所以如果浏览器禁用 cookie 的话，session 的使用也会受到影响。

54.cookie 生成过程

Web 客户端通过浏览器向 Web 服务器发送连接请求；Web 服务器接收到请求后，根据用户端提供的信息产生一个 Set-Cookies Head，并将生成的 Set-Cookies Header 通过 Response Header 存放在 HTTP 报文中回传给 Web 客户端，建立一次会话连接；Web 客户端收到 HTTP 应答报文后，如果要继续已建立的这次会话，则将 Cookies 的内容从 HTTP 报文中取出，形成一个 Cookies 文本文件储存在客户端计算机的硬盘中或保存 在客户端计算机的内存中。
在写入时，Cookie 由三元组【名字 name，域名 domain，路径 path】确定，所以会出现重名，即 cookie 不唯一。例如：
Set-Cookie: user2=bbb; domain=www.bank.com; path=/;

55. cookie 常用参数

key，value 对应保存
max_age; expires 设置过期时间
path，domain，用于匹配
secure，httponly，解决安全问题

56.cookie 植入请求过程

cookie 是放在请求头中的，client 把 cookie 通过 HTTP Request 中的“Cookie: header”发送给 server。
能够发送 cookie 要求：本地有缓存的 cookies，且能根据 url 来匹配 cookie 的 domain，path 属性；且 cookie 尚未过期。

57.cookie 在植入请求时，是如何选择匹配的 cookie 的

匹配原则：
一：domain 向上匹配，一个页面可以为本域和任何父域设置 cookie。
ex：当前页面为 http://www.bank.com
Set-Cookie: user1=aaa; domain=.bank.com; path=/; -------> domain 不配陪，无法向下匹配
Set-Cookie: user2=bbb; domain=www.bank.com; path=/; -------> domain 匹配
Set-Cookie: user3=ccc; domain=.www.bank.com; path=/; ------->domian 匹配，向上（父域）匹配
Set-Cookie: user4=ddd; domain=other.bank.com； path=/; ------->domain 不匹配
二：path 向下匹配，一个页面可以为本路径和任何子路径设置 cookie
ex：当页面为 http://www.bank.com ，path 为/hello
Set-Cookie: user1=aaa; domain=www.bank.com; path=/; -------> path 不向上匹配
Set-Cookie: user1=aaa; domain=www.bank.com; path=/hello; -------> path 匹配
Set-Cookie: user2=bbb; domain=www.bank.com; path=/hello/world; -------> path 向下（子路径）匹配

58.如何处理 cookie 的不唯一问题

应该遵循的优先级为：优先 path 更长的，如果 path 相同，则优先更早创建的 cookie。

59.cookie 在跨域时是如何携带 cookie 的

在解决跨域时，我们会采用很多解决跨域的方法。此处以 cors 为例。
cors 默认不发 cookie。如果要发 cookie，客户端需要设置 withCredentials 属性为 true（withCredentials 属性会包含来自远程域的请求的任何 cookie，但这些 cookie 依然遵循同源策略），需要服务端使用 Access-Control-Allow-Credentials: true 字段来允许携带 cookie。

60.cookie 欺骗

攻击者登录的是受害者的身份
cookie 在 http 协议中是明文传输的，并且直接附在 http 报文的前面，所以只要使用嗅探工具，获取 http 包，就可以分析并获得 cookie 的值（这个时候即使对 cookie 内容加密也没有作用，认为人家不需要看懂，只需要拿来放在请求里面就好了）。此后只要使用这个 cookie 的值加载请求中，就会被服务器当成用户。即使加密 cookie，使用 https，也可能会被攻击者诱使用户访问 http 网站从而获取未加密的 cookie。
因此可以使用 cookie 中的 secure 属性，设置为 true 时，表示创建的 Cookie 会被以安全的形式向服务器传输，也就是只能在 HTTPS 连接中被浏览器传递到服务器端进行会话验证，如果是 HTTP 连接则不会传递该 cookie 信息，所以不会被窃取到 Cookie 的具体内容。
另一个是 HttpOnly 属性，如果在 Cookie 中设置了"HttpOnly"属性，那么通过程序(JS 脚本、Applet 等)将无法读取到 Cookie 信息，这样能有效的防止 XSS 攻击。
区别：secure 属性是防止信息在传递的过程中被监听捕获后信息泄漏，HttpOnly 属性的目的是防止程序获取 cookie 后进行攻击。

61.cookie 注入

认证为攻击者的攻击方式，受害者登录的是攻击者的身份。
要进行 cookie 注入，有两个必须条件： 1 是程序对 get 和 post 方式提交的数据进行了过滤，但未对 cookie 方式提交的数据库进行过滤；2 是在条件 1 的基础上还需要程序对提交数据获取方式是直接 request(“xxx”)的方式，未指明使用 request 对象的具体方法进行获取。
原理：用户先是用 https 的方式访问 bank.com，而后，我们让用户使用 http 的方式来访问 non.bank.com，这时我们能得到一个 cookie ，攻击者此时就可以将该明文 cookie 替换成攻击者 attack 的 cookie。在 domain 向上匹配的同源策

略下和cookie 优先级的情况下,访问non.bank.com时得到的cookie 会更优先，这时用户通过 https 访问 bank.com 时，就会使用 attack 优先级更高的 cookie。

62. 常用那几种浏览器测试？

IE、Safari、Chrome、Mozilla Firefox、Opera

63. 主流浏览器的内核有哪些？

1、Trident 内核
代表产品为 Internet Explorer，又称其为 IE 内核。Trident（又称为 MSHTML），是微软开发的一种排版引擎 。
2、Gecko 内核
代表作品为 Mozilla Firefox。Gecko 是一套开放源代码的、以 C++编写的网页排版引擎，是最流行的排版引擎之一，仅次于 Trident。使用它的最著名浏览器有 Firefox。
3、WebKit 内核
代表作品有 Safari、Chrome。WebKit 是一个开源项目，主要用于 Mac OS 系统，它的特点在于源码结构清晰、渲染速度极快。缺点是对网页代码的兼容性不高，导致一些编写不标准的网页无法正常显示。
4、Presto 内核
代表作品 Opera。Presto 是由 Opera Software 开发的浏览器排版引擎，供 Opera 7.0 及以上使用。

64.说说你对浏览器内核的理解？

浏览器内核主要包括以下三个技术分支：排版渲染引擎、 JavaScript 引擎，以及其他。
排版渲染引擎：主要负责取得网页的内容(HTML、XML、图像等)、整理信息，以及计算网页的显示方式，然后输出至显示器。
JavaScript 引擎：是用来渲染 JavaScript 的，JavaScript 的渲染速度越快，动态网页的展示也越快。

65.URL 和 URI 有什么区别

URI 是统一资源标识符，相当于一个人身份证号码
Web 上可用的每种资源如 HTML 文档、图像、视频片段、程序等都是一个来 URI 来定位的
URI 一般由三部组成
①访问资源的命名机制
②存放资源的主机名
③资源自身的名称，由路径表示，着重强调于资源。
URL 是统一资源定位符，相当于一个人的家庭住址
URL 是 Internet 上用来描述信息资源的字符串，主要用在各种 WWW 客户程序和服务器程序上，特别是著名的 Mosaic。采用 URL 可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。
URL 一般由三部组成
①协议(或称为服务方式)
②存有该资源的主机 IP 地址(有时也包括端口号)
③主机资源的具体地址。如目录和文件名等。

66. HTTP 和 HTTPS 的区别

HTTPS 协议需要到 CA （Certificate Authority，证书颁发机构）申请证书，一般免费证书较少，因而需要一定费用
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。
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。
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。(无状态的意思是其数据包的发送、传输和接收都是相互独立的。无连接的意思是指通信双方都不长久的维持对方的任何信息。)

67.简单说一下浏览器本地存储是怎样的

在较高版本的浏览器中，js 提供了 sessionStorage 和 globalStorage。在 HTML5 中提供了 localStorage 来取代 globalStorage。
html5 中的 Web Storage 包括了两种存储方式：
sessionStorage

localStorage
sessionStorage
sessionStorage 用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此 sessionStorage 不是一种持久化的本地存储，仅仅是会话级别的存储。
localStorage
localStorage 用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

68.请你谈谈关于 Cookie 的利弊

cookie 虽然在持久保存客户端数据提供了方便，分担了服务器存储的负担，但还是有很多局限性的。
第一：每个特定的域名下最多生成 20 个 cookie
IE6 或更低版本最多 20 个 cookie
IE7 和之后的版本最后可以有 50 个 cookie
Firefox 最多 50 个 cookie
chrome 和 Safari 没有做硬性限制
IE 和 Opera 会清理近期最少使用的 cookie，Firefox 会随机清理 cookie。
cookie 的最大大约为 4096 字节，为了兼容性，一般不能超过 4095 字节。
IE 提供了一种存储可以持久化用户数据，叫做 uerData，从 IE5.0 就开始支持。每个数据最多 128K，每个域名下最多 1M。这个持久化数据放在缓存中，如果缓存没有清理，那么会一直存在。
优点：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（SSL），减少 cookie 被破解的可能性。
只在 cookie 中存放不敏感数据，即使被盗也不会有重大损失。
控制 cookie 的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的 cookie。
缺点：
Cookie 数量和长度的限制。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 被人拦截了，那人就可以取得所有的 session 信息。即使加密也与事无补，因为拦截者并不需要知道 cookie 的意义，他只要原样转发 cookie 就可以达到目的了。
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，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

69.为什么 JavaScript 是单线程的，与异步冲突吗

补充：JS 中其实是没有线程概念的，所谓的单线程也只是相对于多线程而言。JS 的设计初衷就没有考虑这些，针对 JS 这种不具备并行任务处理的特性，我们称之为“单线程”。
JS 单线程是指一个浏览器进程中只有一个 JS 的执行线程，同一时刻内只会有一段代码在执行。
举个通俗例子，假设 JS 支持多线程操作的话，JS 可以操作 DOM，那么一个线程在删除 DOM，另外一个线程就在获取 DOM 数据，这样子明显不合理，这算是证明之一。
来看段代码

```
function foo() {    console.log("first");
    setTimeout(( function(){        console.log( 'second' );
    }),5);
}
for (var i = 0; i < 1000000; i++) {
    foo();
}复制代码
```

打印结果就是首先是很多个 first,然后再是 second。
异步机制是浏览器的两个或以上常驻线程共同完成的，举个例子，比如异步请求由两个常驻线程，JS 执行线程和事件触发线程共同完成的。
JS 执行线程发起异步请求（浏览器会开启一个 HTTP 请求线程来执行请求，这时 JS 的任务完成，继续执行线程队列中剩下任务）
然后在未来的某一时刻事件触发线程监视到之前的发起的 HTTP 请求已完成，它就会把完成事件插入到 JS 执行队列的尾部等待 JS 处理
再比如定时器触发(settimeout 和 setinterval) 是由「浏览器的定时器线程」执行的定时计数，然后在定时时间把定时处理函数的执行请求插入到 JS 执行队列的尾端(所以用这两个函数的时候,实际的执行时间是大于或等于指定时间的，不保证能准确定时的)。
所以这么说，JS 单线程与异步更多是浏览器行为，之间不冲突。

70. CSS 加载会造成阻塞吗

先给出结论
CSS 不会阻塞 DOM 解析，但会阻塞 DOM 渲染。
CSS 会阻塞 JS 执行，并不会阻塞 JS 文件下载
先讲一讲 CSSOM 作用
第一个是提供给 JavaScript 操作样式表的能力
第二个是为布局树的合成提供基础的样式信息
这个 CSSOM 体现在 DOM 中就是 document.styleSheets。

由之前讲过的浏览器渲染流程我们可以看出：
DOM 和 CSSOM 通常是并行构建的，所以「CSS 加载不会阻塞 DOM 的解析」。
然而由于 Render Tree 是依赖 DOM Tree 和 CSSOM Tree 的，所以它必须等到两者都加载完毕后，完成相应的构建，才开始渲染，因此，「CSS 加载会阻塞 DOM 渲染」。
由于 JavaScript 是可操纵 DOM 和 css 样式 的,如果在修改这些元素属性同时渲染界面（即 JavaScript 线程和 UI 线程同时运行）,那么渲染线程前后获得的元素数据就可能不一致了。
因此为了防止渲染出现不可预期的结果,浏览器设置 「GUI 渲染线程与 JavaScript 引擎为互斥」的关系。
有个需要注意的点就是：
「有时候 JS 需要等到 CSS 的下载，这是为什么呢？」
仔细思考一下，其实这样做是有道理的，如果脚本的内容是获取元素的样式，宽高等 CSS 控制的属性，浏览器是需要计算的，也就是依赖于 CSS。浏览器也无法感知脚本内容到底是什么，为避免样式获取，因而只好等前面所有的样式下载完后，再执行 JS。
JS 文件下载和 CSS 文件下载是并行的，有时候 CSS 文件很大，所以 JS 需要等待。
因此,样式表会在后面的 js 执行前先加载执行完毕,所以「css 会阻塞后面 js 的执行」。

71.为什么 JS 会阻塞页面加载

「JS 阻塞 DOM 解析」，也就会阻塞页面
这也是为什么说 JS 文件放在最下面的原因，那为什么会阻塞 DOM 解析呢
你可以这样子理解：
由于 JavaScript 是可操纵 DOM 的,如果在修改这些元素属性同时渲染界面(即 JavaScript 线程和 UI 线程同时运行),那么渲染线程前后获得的元素数据就可能不一致了。
因此为了防止渲染出现不可预期的结果,浏览器设置 「GUI 渲染线程与 JavaScript 引擎为互斥」的关系。
当 JavaScript 引擎执行时 GUI 线程会被挂起,GUI 更新会被保存在一个队列中等到引擎线程空闲时立即被执行。
当浏览器在执行 JavaScript 程序的时候,GUI 渲染线程会被保存在一个队列中,直到 JS 程序执行完成,才会接着执行。
因此如果 JS 执行的时间过长,这样就会造成页面的渲染不连贯,导致页面渲染加载阻塞的感觉。
另外，如果 JavaScript 文件中没有操作 DOM 相关代码，就可以将该 JavaScript 脚本设置为异步加载，通过 async 或 defer 来标记代码。

72.defer 和 async 的区别 ？

两者都是异步去加载外部 JS 文件，不会阻塞 DOM 解析。
Async 是在外部 JS 加载完成后，浏览器空闲时，Load 事件触发前执行，标记为 async 的脚本并不保证按照指定他们的先后顺序执行，该属性对于内联脚本无作用 (即没有「src」属性的脚本)。
defer 是在 JS 加载完成后，整个文档解析完成后，触发 DOMContentLoaded 事件前执行，如果缺少 src 属性（即内嵌脚本），该属性不应被使用，因为这种情况下它不起作用。

73. DOMContentLoaded 与 load 的区别 ？

DOMContentLoaded 事件触发时：仅当 DOM 解析完成后，不包括样式表，图片等资源。
onload 事件触发时,页面上所有的 DOM,样式表,脚本,图片等资源已经加载完毕。
那么也就是先 DOMContentLoaded -> load, 那么在 Jquery 中， 使用 (document).load(callback)监听的就是 load 事件。
那我们可以聊一聊它们与 async 和 defer 区别
带 async 的脚本一定会在 load 事件之前执行，可能会在 DOMContentLoaded 之前或之后执行。
情况 1：HTML 还没有被解析完的时候，async 脚本已经加载完了，那么 HTML 停止解析，去执行脚本，脚本执行完毕后触发 DOMContentLoaded 事件
情况 2：HTML 解析完了之后，async 脚本才加载完，然后再执行脚本，那么在 HTML 解析完毕、async 脚本还没加载完的时候就触发 DOMContentLoaded 事件
如果 script 标签中包含 defer，那么这一块脚本将不会影响 HTML 文档的解析，而是等到 HTML 解析完成后才会执行。而 DOMContentLoaded 只有在 defer 脚本执行结束后才会被触发。
情况 1：HTML 还没解析完成时，defer 脚本已经加载完毕，那么 defer 脚本将等待 HTML 解析完成后再执行。defer 脚本执行完毕后触发 DOMContentLoaded 事件
情况 2：HTML 解析完成时，defer 脚本还没加载完毕，那么 defer 脚本继续加载，加载完成后直接执行，执行完毕后触发 DOMContentLoaded 事件。

74.为什么 CSS 动画比 JavaScript 高效

我觉得这个题目说法上可能就是行不通，不能这么说，如果了解的话，都知道 will-change 只是一个优化的手段，使用 JS 改变 transform 也可以享受这个属

性带来的变化，所以这个说法上有点不妥。
所以围绕这个问题展开话，更应该说建议推荐使用 CSS 动画，至于为什么呢，涉及的知识点大概就是重排重绘，合成，这方面的点，我在浏览器渲染流程中也提及了。
尽可能的避免重排和重绘，具体是哪些操作呢，如果非要去操作 JS 实现动画的话，有哪些优化的手段呢？
比如
使用 createDocumentFragment 进行批量的 DOM 操作
对于 resize、scroll 等进行防抖/节流处理。
rAF 优化等等
剩下的东西就留给你们思考吧，希望我这是抛砖引玉吧。

75.高性能动画是什么，那它衡量的标准是什么呢？

动画帧率可以作为衡量标准，一般来说画面在 60fps 的帧率下效果比较好。
换算一下就是，每一帧要在 16.7ms (16.7 = 1000/60) 内完成渲染。
我们来看看 MDN 对它的解释吧
window.requestAnimationFrame() 方法告诉浏览器您希望执行动画并请求浏览器在下一次重绘之前调用指定的函数来更新动画。该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。— MDN
当我们调用这个函数的时候，我们告诉它需要做两件事：
我们需要新的一帧；
当你渲染新的一帧时需要执行我传给你的回调函数。

76. rAF 与 setTimeout 相比

rAF(requestAnimationFrame) 最大的优势是「由系统来决定回调函数的执行时机」。
具体一点讲就是，系统每次绘制之前会主动调用 rAF 中的回调函数，如果系统绘制率是 60Hz，那么回调函数就每 16.7ms 被执行一次，如果绘制频率是 75Hz，那么这个间隔时间就变成了 1000/75=13.3ms。
换句话说就是，rAF 的执行步伐跟着系统的绘制频率走。它能保证回调函数在屏幕每一次的绘制间隔中只被执行一次(上一个知识点刚刚梳理完「函数节流」)，这样就不会引起丢帧现象，也不会导致动画出现卡顿的问题。
另外它可以自动调节频率。如果 callback 工作太多无法在一帧内完成会自动降低为 30fps。虽然降低了，但总比掉帧好。
与 setTimeout 动画对比的话，有以下几点优势
当页面隐藏或者最小化时，setTimeout 仍然在后台执行动画，此时页面不可见或者是不可用状态，动画刷新没有意义，而且浪费 CPU。

rAF 不一样，当页面处理未激活的状态时，该页面的屏幕绘制任务也会被系统暂停，因此跟着系统步伐走的 rAF 也会停止渲染，当页面被激活时，动画就从上次停留的地方继续执行，有效节省了 CPU 开销。

77.什么时候调用呢

规范中似乎是这么去定义的：
在重新渲染前调用。
很可能在宏任务之后不去调用
这样子分析的话，似乎很合理嘛，为什么要在重新渲染前去调用呢？因为 rAF 作为官方推荐的一种做流畅动画所应该使用的 API，做动画不可避免的去操作 DOM，而如果是在渲染后去修改 DOM 的话，那就只能等到下一轮渲染机会的时候才能去绘制出来了，这样子似乎不合理。
rAF 在浏览器决定渲染之前给你最后一个机会去改变 DOM 属性，然后很快在接下来的绘制中帮你呈现出来，所以这是做流畅动画的不二选择。
至于宏任务，微任务，这可以说起来就要展开篇幅了，暂时不在这里梳理了。

78.rAF 与节流相比

跟 _.throttle(dosomething, 16) 等价。它是高保真的，如果追求更好的精确度的话，可以用浏览器原生的 API 。
可以使用 rAF API 替换 throttle 方法，考虑一下优缺点：
优点
动画保持 60fps（每一帧 16 ms），浏览器内部决定渲染的最佳时机
简洁标准的 API，后期维护成本低
缺点
动画的开始/取消需要开发者自己控制，不像 ‘.debounce’ 或 ‘.throttle’ 由函数内部处理。
浏览器标签未激活时，一切都不会执行。
尽管所有的现代浏览器都支持 rAF ，IE9，Opera Mini 和 老的 Android 还是需要打补丁。
Node.js 不支持，无法在服务器端用于文件系统事件。
根据经验，如果 JavaScript 方法需要绘制或者直接改变属性，我会选择 requestAnimationFrame，只要涉及到重新计算元素位置，就可以使用它。
涉及到 AJAX 请求，添加/移除 class （可以触发 CSS 动画），我会选择 _.debounce 或者 _.throttle ，可以设置更低的执行频率（例子中的 200ms 换成 16ms）。

前端面试题集锦——服务端与网络

1. http/https 协议

1.0 协议缺陷:
o 无法复用链接，完成即断开，重新慢启动和 TCP 3 次握手
ohead of line blocking: 线头阻塞，导致请求之间互相影响
1.1 改进:
o 长连接(默认 keep-alive)，复用
ohost 字段指定对应的虚拟站点
o 新增功能:
断点续传
身份认证
状态管理
cache 缓存
Cache-Control
Expires
Last-Modified
Etag
2.0:
o 多路复用
o 二进制分帧层: 应用层和传输层之间
o 首部压缩
o 服务端推送
https: 较为安全的网络传输协议
o 证书(公钥)
oSSL 加密
o 端口 443
TCP:
o 三次握手
o 四次挥手
o 滑动窗口: 流量控制
o 拥塞处理
慢开始
拥塞避免
快速重传
快速恢复
缓存策略: 可分为 强缓存 和 协商缓存
Cache-Control/Expires: 浏览器判断缓存是否过期，未过期时，直接使用强缓存，Cache-Control

的 max-age 优先级高于 Expires
当缓存已经过期时，使用协商缓存
唯一标识方案: Etag(response 携带) & If-None-Match(request 携带，上一次返回的 Etag): 服务器判断资源是否被修改，
最后一次修改时间: Last-Modified(response) & If-Modified-Since (request，上一次返回的 Last-Modified)
如果一致，则直接返回 304 通知浏览器使用缓存
如不一致，则服务端返回新的资源
Last-Modified 缺点：
周期性修改，但内容未变时，会导致缓存失效
最小粒度只到 s， s 以内的改动无法检测到
Etag 的优先级高于 Last-Modified

2. 常见状态码

1xx: 接受，继续处理
200: 成功，并返回数据
201: 已创建
202: 已接受
203: 成为，但未授权
204: 成功，无内容
205: 成功，重置内容
206: 成功，部分内容
301: 永久移动，重定向
302: 临时移动，可使用原有 URI
304: 资源未修改，可使用缓存
305: 需代理访问
400: 请求语法错误
401: 要求身份认证
403: 拒绝请求
404: 资源不存在
500: 服务器错误

3. get / post

get: 缓存、请求长度受限、会被历史保存记录
o 无副作用(不修改资源)，幂等(请求次数与资源无关)的场景
post: 安全、大数据、更多编码类型
两者详细对比如下图:

4. Websocket

Websocket 是一个 持久化的协议， 基于 http ， 服务端可以 主动 push
兼容：
oFLASH Socket
o 长轮询： 定时发送 ajax
olong poll： 发送 --> 有消息时再 response
new WebSocket(url)
ws.onerror = fn
ws.onclose = fn
ws.onopen = fn
ws.onmessage = fn
ws.send()

5. TCP 三次握手

建立连接前，客户端和服务端需要通过握手来确认对方:
客户端发送 syn(同步序列编号) 请求，进入 syn_send 状态，等待确认
服务端接收并确认 syn 包后发送 syn+ack 包，进入 syn_recv 状态
客户端接收 syn+ack 包后，发送 ack 包，双方进入 established 状态

6. TCP 四次挥手

客户端 -- FIN --> 服务端， FIN—WAIT
服务端 -- ACK --> 客户端， CLOSE-WAIT
服务端 -- ACK,FIN --> 客户端， LAST-ACK
客户端 -- ACK --> 服务端，CLOSED

7. Node 的 Event Loop: 6 个阶段

timer 阶段: 执行到期的 setTimeout / setInterval 队列回调
I/O 阶段: 执行上轮循环残流的 callback
idle, prepare

poll: 等待回调
1.执行回调
1.执行定时器
如有到期的 setTimeout / setInterval， 则返回 timer 阶段
如有 setImmediate，则前往 check 阶段
check
o 执行 setImmediate
close callbacks

8.URL 概述

URL 是统一资源定位符的简称，也就是说根据 URL 能够定位到网络上的某个资源，它是指向互联网“资源”的指针。
每个 URL 都是 URI，但不一定每个 URI 都是 URL，这是因为 URI 还包括一个子类，即统一资源名称（URN），它命名资源但不指定如何定位资源。URL 是统一资源定位符，是对可以从互联网上得到的资源的位置和访问方法的一种简洁的表示，是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的 URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。比如百度 URL 即是 http://www.baidu.com。

9.安全

XSS 攻击: 注入恶意代码
ocookie 设置 httpOnly
o 转义页面上的输入内容和输出内容
CSRF: 跨站请求伪造，防护:
oget 不修改数据
o 不被第三方网站访问到用户的 cookie
o 设置白名单，不被第三方网站请求
o 请求校验

10. HTTPS 和 HTTP 的区别

HTTPS 协议需要到 CA（证书颁发机构）申请证书（公钥），一版免费证书很少，需要交费；HTTP 协议运行在 TCP 之上，所有的传输都是明文；HTTPS 运行在 SSL/TSL 之上，SSL/TLS 运行在 TCP 之上，所有传输的内容都经过加密的。
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。
http 的连接很简单，是无状态的；HTTPS 协议是由 HTTP + SSL 协议构建的可进行加密传输、

身份认证的网络协议，可以有效的防止运营商劫持，解决了防劫持的一个大问题，比 http 协议安全。

11. HTTP 版本

HTTP/1.0

最早的 http 只是使用一些简单的网页上和网络请求上，每次请求都打开一个新的 TCP 连接，收到响应后立即断开连接

HTTP/1.1

缓存处理，HTTP/1.1 更多的引入了缓存策略，如 Cache-Control，Entity tag，If-Unmodified-Since, If-Match, If-None-Match 等

宽带优化及网络连接的使用，在 HTTP/1.0 中，存在一些浪费宽带的现象，列如客户端只需要某个对象的一部分，而服务器把整个对象都送过来了，并且不支持断点续传，HTTP1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

错误通知的管理，在 HTTP/1.1 中新增了 24 个错误状态响应码，如 409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

Host 头处理，在 HTTP1.0 中认为每台服务器都绑定一个唯一的 IP 地址，因此，请求消息中的 URL 并没有传递主机名。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个 IP 地址。HTTP1.1 的请求消息和响应消息都应支持 Host 头域，且请求消息中如果没有 Host 头域会报告一个错误（400 Bad Request）

长连接， HTTP/1.1 默认开启持久连接（默认：keep-alive），在一个 TCP 连接上可以传递多个 HTTP 请求和响应，减少了建立与关闭连接的消耗和延迟

HTTP/2.0

在 HTTP/2.0 中，有两个重要的概念，分别是帧（frame） 和 流（stream），帧代表数据传输的最小单位，每个帧都有序列标识标明该帧属于哪个流，流也就是多个帧组成的数据流，每个流表示一个请求。

新的二进制格式： HTTP/1.x 的解析是基于文本的。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认 0 和 1 的组合。基于这种考虑 HTTP2.0 的协议解析决定采用二进制格式，实现方便且健壮。

多路复用： HTTP/2.0 支持多路复用，这是 HTTP/1.1 持久连接的升级版。多路复用，就是在一个 TCP 连接中存在多个条流，也就是多个请求，服务器则可以通过帧中的标识知道该帧属于哪个流（即请求），通过重新排序还原请求。多路复用允许并发多个请求，每个请求及该请求的响应不需要等待其他的请求或响应，避免了线头阻塞问题。这样某个请求任务耗时严重，不会影响到其它连接的正常执行,极大的提高传输性能。

头部压缩： 对前面提到的 HTTP/1.x 的 header 带有大量信息，而且每次都要重复发送，HTTP/2.0 使用 encoder 来减少需要传输的头部大小，通讯双方各自 cache 一份头部 fields 表，既避免了重复头部的传输，又减小了需要传输的大小。

服务端推送： 服务端推送指把客户端所需要的 css/js/img 资源伴随着 index.html 一起发送到客户端，省去了客户端重复请求的步骤（从缓存中取）。正因为没有发起请求，建立连接等操作，所以静态资源通过服务端推送的方式极大的提升了速度

HTTP/3.0

HTTP/2.0 使用了多路复用，一般来说同一域名下只需要使用一个 TCP 连接。但当这个连接中出现了丢包的情况，会导致整个 TCP 都要开始等待重传，也就导致了后面所有的数据都阻塞了。

避免包阻塞： 多个流的数据包在 TCP 连接上传输时，若一个流中的数据包传输出现问题，TCP 需要等待该包重传后，才能继续传输其它流的数据包。但在基于 UDP 的 QUIC 协议中，不同的流之间的数据传输真正实现了相互独立互不干扰，某个流的数据包在出问题需要重传时，并不会对其他流的数据包传输产生影响。

快速重启会话: 普通基于 tcp 的连接，是基于两端的 ip 和端口和协议来建立的。在网络切换场景，例如手机端切换了无线网，使用 4G 网络，会改变本身的 ip，这就导致 tcp 连接必须重新创建。而 QUIC 协议使用特有的 UUID 来标记每一次连接，在网络环境发生变化的时候，只要 UUID 不变，就能不需要握手，继续传输数据。

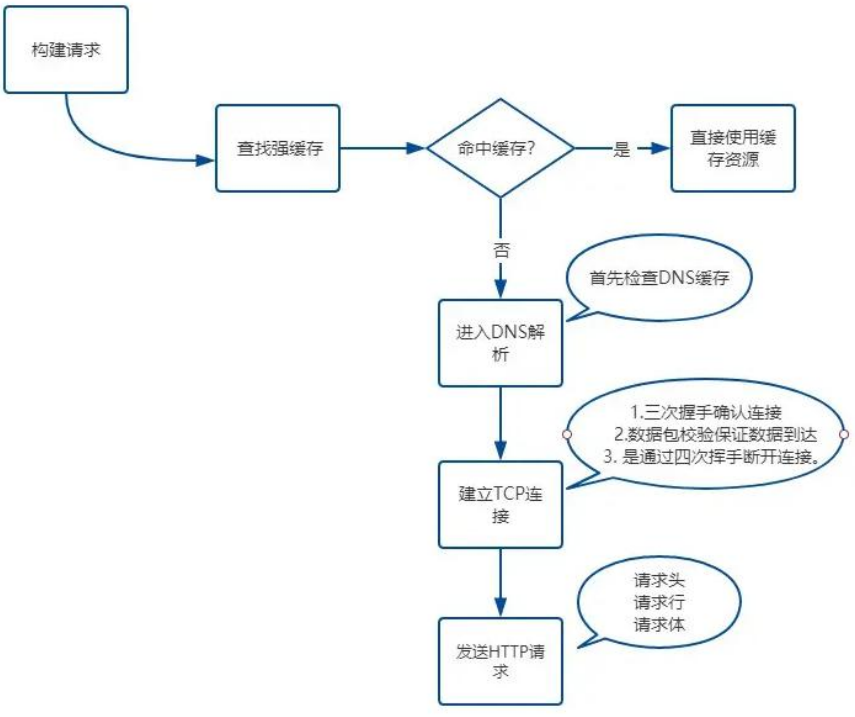
HTTP2.0 的多路复用和 HTTP1.X 中的长连接有什么区别？

HTTP/1.* 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；

HTTP/1.1 在一个 TCP 连接上可以传递多个 HTTP 请求和响应，后面的请求等待前面的请求返回才能获得执行机会，一旦有某个请求超时，后续请求只能被阻塞，毫无办法，也就是常说的线头阻塞

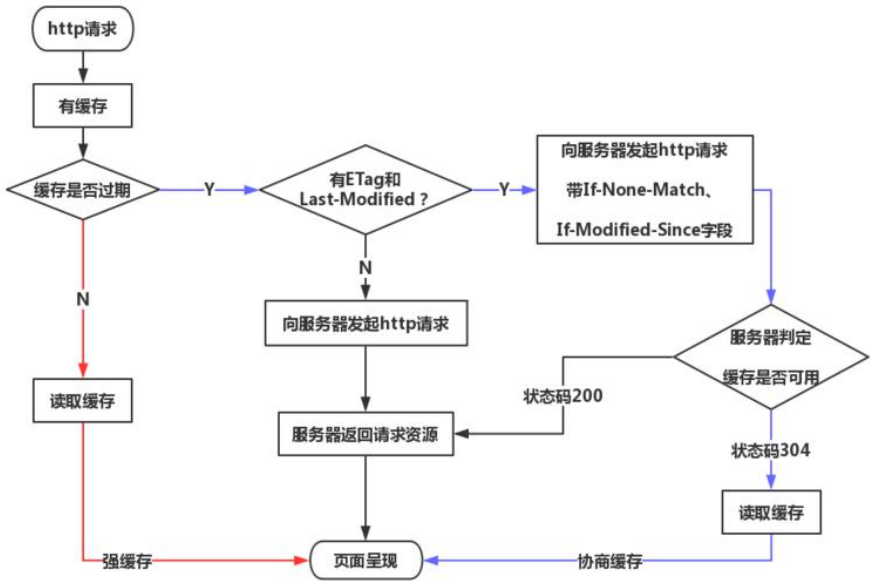
HTTP/2.0 多个请求可同时在一个连接上并行执行.某个请求任务耗时严重，不影响其他连接的正常执行。

12. 从输入 URL 到页面呈现发生了什么？



13. HTTP 缓存

浏览器第一次向一个 web 服务器发起请求的时候，服务器会返回请求的资源，并依据返回的响应头字段来告诉如何缓存，一般分为强缓存和协商缓存两种。当再次发出请求时候就可以依据缓存策略来进行读取资源，具体过程如下图：



14.缓存位置

当强缓存命中或者协商缓存中服务器返回 304 的时候，我们直接从缓存中获取资源。那么资源存放在哪里呢

浏览器中的缓存一共分为 4 种，优先级从高到低分别是：

Service Worker

Memory Cache

Disk Cache

Push Cache

Service Worker 即让 JS 运行在主线程之外，由于脱离了浏览器窗口，因此无法访问 DOM. 虽然如此。 但它仍然能帮助我们完成很多有用的功能，比如离线缓存、消息推送和网络代理等功能。其中的离线缓存就是 Service Worker Cache。

Service Worker 同时也是 PWA (Progressive Web App 渐进式增强 WEB 应用) 的重要实现机制。

Memory Cache 内存缓存，从效率上讲它最快. 但是从存活时间上讲又是最短的. 当渲染进程结束后，内存缓存也就不存在了

Disk Cache 磁盘缓存，从存取效率上讲是比内存缓存慢的，但是他的优势在于存储容量和存储时长。

如何决定将资源放入硬盘还是内存？
比较大的 JS、css 文件会直接被丢进磁盘，反之丢进内存
内存使用率较高的时候，文件优先进入磁盘
Push Cache 即推送缓存，是浏览器缓存的最后一道防线。它是 HTTP/2 中的内容，虽然现在应用的并不广泛，但随着 HTTP/2 的推广，它的应用越来越广泛。

15.强缓存

强缓存简介
强缓存是利用 http 头中的 Expries 和 Cache-Control 两个字段进行控制。 强缓存中，当请求再次发出时，浏览器会根据其中的 expires 和 cache-control 判断目标资源是否“命中”强缓存，若命中则直接从缓存中获取资源，不会再与服务端发生通信。
命中强缓存，返回的状态为 200(from disk cache)。 上图中红色的线就是整个流程。
强缓存的实现：从 expires 到 cache-control
实现强缓存，过去一直使用的是 expires。当服务器返回响应的时候，在 Request Headers 中将过期时间写入 expires 字段
expires: Tue, 15 Oct 2019 13:30:54 GMT
exprires 是一个时间戳，当我们再次向服务器请求资源的时候，浏览器就会先对比本地时间和 expires 的时间戳，如果本地时间小于 expires 设定的过期时间，那么就直接去缓存中取这个资源。
当本地时间和服务器上的时间不一致的时候，或者手动改掉本地时间的时候，expires 可能就无法达到我们预期的效果
在 HTTP1.1 中新增了 Cache-Control 字段，通过 max-age 来控制资源的有效期，max-age 是一个相对时间，可以很好的规避 expires 这种时间戳设置带来的问题
cache-control: max-age=31536000
上面表示资源在 31536000 秒以内都是有效的
缓存控制 cache-control
1、no-store 和 no-cahe
no-store 表示不进行缓存，缓存中不得存储任何关于客户端请求和服务端响应的内容。每次由客户端发起的请求都会下载完整的响应内容。
Cache-control: no-store
no-cache 表示不缓存过期的资源，缓存会向源服务器进行有效期确认后处理资源，也许称为 do-notserve-from-cache-without-revalidation 更合适。浏览器默认开启的是 no-cahce，其实这里也可理解为开启协商缓存
Cache-control: no-cache
3、public 和 private
public 与 private 是针对资源是否能够被代理服务缓存而存在的一组对立概念
当我们为资源设置了 pubile，那么它既可以被浏览器缓存也可被代理服务器缓存。设置为 private 的时候，则该资源只能被浏览器缓存，其中默认值是 private。
4、max-age 和 s-maxage
s-maxage 只适用于供多用户使用的公共服务器上(如 CND cache)，并只对 public 缓存有效，客户端中我们只考虑用 max-age.

s-maxage 优先级高于 max-age，两者同时出现时，优先考虑 s-maxage。如果 s-maxage 未过期，则向代理服务器请求其缓存内容
cache-control: max-age=3600, s-maxage=31536000

16.协商缓存

当浏览器发现缓存过期后，缓存并非不可以使用了，因为服务器的资源可能还没发生改变，需要与服务器协商，让服务器判断本地缓存是否还可以使用，具体流程图如上图的蓝线部分。
1、ETag 和 If-None-Match
二者的值都是服务器为每份资源分配的唯一标识字符串。
• 浏览器请求资源，服务器会在响应报文头中加入 ETag 字段。资源更新的时候，服务端的 ETag 值也随之更新
• 浏览器再次请求资源，会在请求报文头中添加 If-None-Match 字段，它的值就是上次响应报文中的 ETag 值，服务器会对比 ETag 和 If-None-Match 的值是否一致。如果不一致，服务器则接受请求，返回更新后的资源，状态码返回 200；如果一致，表明资源未更新，则返回状态码 304，可继续使用本地缓存，值得注意的是此时响应头会加上 ETag 字段，即使它没有变化
2、Last-Modified 和 If-Modified-Since
二者的值都是 GMT 格式的时间字符串
• 浏览器第一次向服务器更新资源后，服务器会在响应头中添加 Last-Modified 字段，表明该资源最后一次的修改时间
• 浏览器再次请求该资源时，会在请求头报文中添加 If-Modifed-Since 字段，他的值就是上次服务器响应报文中 Last-Modified 的值。服务器会对比 Last-Modified 和 If-Modifed-Since 的值是否一致。如果不一致，服务器则接受请求，返回更新后的资源；如果一致，则返回状态为 304 的响应，可继续在本地缓存，与 ETag 不同的是，此时响应头中不会再添加 Last-Modifed 字段
3、ETag 对比 Last-Modified 的优势
HTTP1.1 中的加入 ETag 字段主要为了解决 Last-Modified 的几个问题
• 一些文件也许会周期性的更改，但是他的内容并不改变(仅仅改变的修改时间)，这个时候我们并不希望客户端认为这个文件被修改了，而重新获取
• 某些文件修改非常频繁，比如在秒以下的时间内进行修改，(比方说 1s 内修改了 N 次)，If-Modified-Since 可查到的是秒级，这种修改无法判断。
利用 ETag 可以更加精准的控制缓存，因为 ETag 是服务器自动生成的资源在服务器端的唯一标识符，资源每次变动都会生成新的 ETag 值。Last-Modified 与 ETag 可以同时出现，但服务器会优先验证 ETag

17.缓存的资源在那里

三级缓存原理

- 先在内存中查找,如果有,直接加载(memory cache)
- 如果内存中不存在,则在硬盘中查找,如果有直接加载(disk cache)
- 如果硬盘中也没有,那么就进行网络请求
- 请求获取的资源缓存到硬盘和内存

一般脚本、字体、图片会存在内存当中，非脚本文件如 css 一般缓存在 disk cache 中
由于 CSS 文件加载一次就可渲染出来,我们不会频繁读取它,所以它不适合缓存到内存中，但是 js 之类的脚本却随时可能会执行,如果脚本在磁盘当中,我们在执行脚本的时候需要从磁盘取到内存中来,这样 IO 开销就很大了,有可能导致浏览器失去响应。

18. 用户行为对浏览器缓存的影响

用户操作	Expires/Cache-Control	Last-Modified/Etag
地址栏回车	有效	有效
页面链接跳转	有效	有效
新开窗口	有效	有效
前进、后退	有效	有效
F5刷新	无效	有效
Ctrl+F5刷新	无效	无效

19.缓存的优点

减少了冗余的数据传递，节省宽带流量
减少了服务器的负担，大大提高了网站性能
加快了客户端加载网页的速度 这也正是 HTTP 缓存属于客户端缓存的原因

20.不同刷新的请求执行过程

浏览器地址栏中写入 URL，回车
浏览器发现缓存中有这个文件了，不用继续请求了，直接去缓存拿。（最快）
F5
F5 就是告诉浏览器，别偷懒，好歹去服务器看看这个文件是否有过期了。于是浏览器就胆胆襟襟的发送一个请求带上 If-Modify-since。
Ctrl+F5
告诉浏览器，你先把你缓存中的这个文件给我删了，然后再去服务器请求个完整的资源文件下来。于是客户端就完成了强行更新的操作.

21.为什么会有跨域问题

浏览器同源策略
跨域是浏览器的限制，抓包工具能获取到数据
浏览器发现 AJAX 请求跨域，会自动添加一些附加头信息，有时会多出一次请求，但用户不会有感觉。
导致跨域情况

URL	说明	是否允许通信
http://www.a.com/a.js http://www.a.com/b.js	同一域名下	允许
http://www.a.com/lab/a.js http://www.a.com/script/b.js	同一域名下不同文件夹	允许
http://www.a.com:8000/a.js http://www.a.com/b.js	同一域名，不同端口	不允许
http://www.a.com/a.js https://www.a.com/b.js	同一域名，不同协议	不允许
http://www.a.com/a.js http://70.32.92.74/b.js	域名和域名对应ip	不允许
http://www.a.com/a.js http://script.a.com/b.js	主域相同，子域不同	不允许
http://www.a.com/a.js http://a.com/b.js	同一域名，不同二级域名（同上）	不允许（cookie这种情况下也不允许访问）
http://www.cnblogs.com/a.js http://www.a.com/b.js	不同域名	不允许

总结：不同协议、不同域名（主域、子域）、不同端口都会导致跨域

22.如何解决跨域

jsonp

nginx 反向代理
CORS（跨域资源共享）

23.访问控制场景（简单请求与非简单请求）

简单请求
使用方法之一：
GET
POST
HEAD
请求的 header
Accept
Accept-Language
Content-Language
Content-Type
text/plain
multipart/form-data
application/x-www-form-urlencoded
使用 Origin 和 Access-Control-Allow-Origin 就能完成最简单的访问控制。
服务端返回的 Access-Control-Allow-Origin:* 表明，该资源可以被任意外域访问
非简单请求
非简单请求对服务的要求不同，比如请求方法为 PUT、DELETE 。 或者 Content-Type 为 application/json
1. 预检请求
使用 OPTIONS 方法，询问。预检请求需要三个字段
Access-Control-Request-Method 告知服务器，实际请求使用的方法
Access-Control-Request-Headers 告知服务器，实际请求携带的请求首部字段
Origin 表示请求来自的那个域
2.预检响应
服务器收到预检请求后，检查了 Origin，Access-Control-Request-Method，Access-Control-Request-Headers 字段后，确认允许跨域，就可以做出回应
如果浏览器否认了“预检”请求，会返回一个正常的 HTTP 回应，但是没有任何 CORS 相关的头部信息字段，浏览器会认为不同意，触发一个错误
服务器回应的其他 CORS 字段
Access-Control-Allow-Methods: 必需，逗号分隔的字符串，表示服务器支持的所有跨域请求方法；
Access-Control-Allow-Headers：浏览器支持的所有头部字段；
Access-Control-Allow-Credentials：true
Access-Control-Allow-Max-Age: 指定本次请求的有效期。

24.withCredentials 属性

CORS 默认不发送 Cookie 和 HTTP 认证信息，如果要把 Cookie 发送到服务器，一方面需要服务器同意，设置响应头 Access-Control-Allow-Credentials：true；另一方面，客户端发送请求时，也要进行一些设置，例如 xhr.withCredentials = true。

25.服务器如何设置 CORS

如果请求为简单请求
直接设置响应头 Access-Control-Allow-Origin 为 *，或者具体的域名；
如果响应头 Access-Control-Allow-Credentials 为 true，则此时 Access-Control-Allow-Origin 不能设置为*，必须指定明确的域名；
如果请求为非简单请求
处理 OPTIONS 请求，服务端可以单独写一个路由
可以把这部分抽离处理，作为一个中间件，例如 Koa

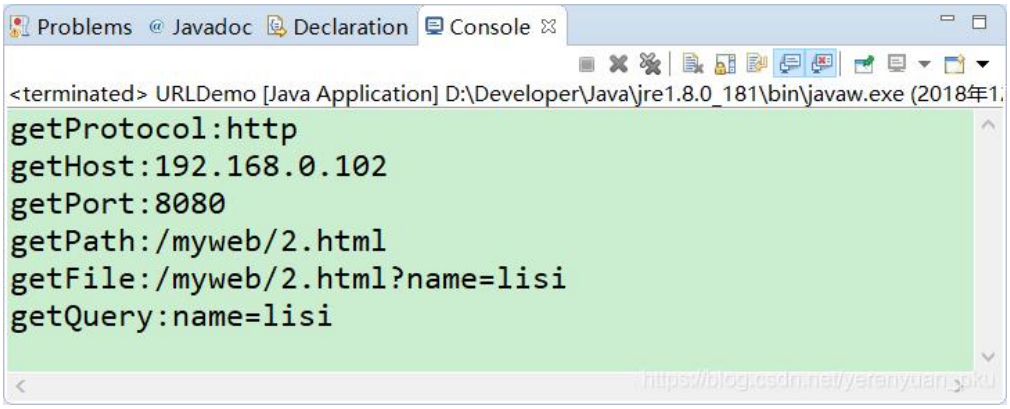
26.URL 类中的常用方法

此类必须掌握以下方法：
方法声明　功能描述
String getFile()　获取此 URL 的文件名
String getHost()　获取此 URL 的主机名（如果适用）
String getPath()　获取此 URL 的路径部分
int getPort() 获取此 URL 的端口号。如果未设置端口号，则返回-1
String getProtocol() 获取此 URL 的协议名称
String getQuery() 获取此 URL 的查询部分

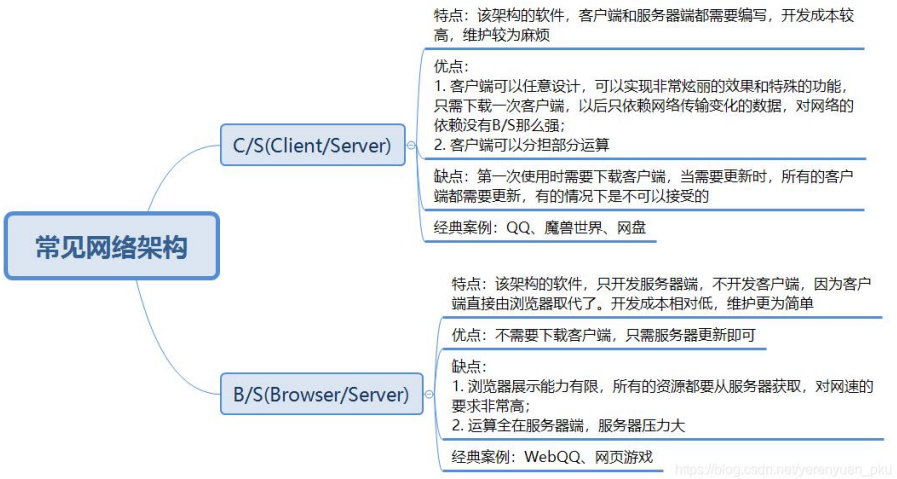
```
package cn.liayun.net.url;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
public class URLDemo {
    public static void main(String[] args) throws IOException {
        //解析 URL 中的数据，使用 URL 对象。
```

```
        String                          str_url                          =
"http://192.168.0.102:8080/myweb/2.html?name=lisi";

        URL url = new URL(str_url);

        System.out.println("getProtocol:" + url.getProtocol());
        System.out.println("getHost:" + url.getHost());
        System.out.println("getPort:" + url.getPort());
        System.out.println("getPath:" + url.getPath());
        System.out.println("getFile:" + url.getFile());
        System.out.println("getQuery:" + url.getQuery());
    }

}
```

运行结果如下：



27. 常见网络架构



28.TCP 连接过程客户端和服务端状态

开始前，客户端和服务端都是关闭的
客户端：closed 状态
服务端：closed 状态
服务器创建 socket 后开始监听
服务端：listen 状态
客户端请求建立 TCP 连接，向服务端发送 SYN 报文
客户端：SYN_SEND 状态
服务器收到客户端的报文后向客户端发送 ACK 和 SYN 报文
服务端：SYN_RCVD 状态
客户端收到 ACK 和 SYN 后向服务器再回一个 ACK
客户端：ESTABLISHED 状态
服务端收到客户端的 ACK 后变为 ESTABLISHED 状态
服务端：ESTABLISHED 状态

29.多进程多线程的区别

1.进程是资源调度的最小单位，线程是 cpu 调度的最小单位
2.数据共享和同步：多进程（数据共享复杂，需要 ipc；数据是分开的，同步简单）多线程（共享进程数据，数据同步复杂）
3.内存，cpu： 多进程（占用内存多，切换复杂，cpu 利用率低）多线程（占用内存少，切换简单，cpu 利用率高）
4.编程调试：多进程（编程调试简单）多线程（变成调试复杂）
5.可靠性：多进程（进程间不会互相影响）多线程（一个线程挂掉导致整个进程挂掉）
6.分布式：多进程（适应于多核，多机分布式；如果一台机器不够，扩展到多台机器比较简单）多线程（适应于多核分布式）

30.OSI，TCP/IP，五层协议的体系结构，以及各层协议

OSI 分层 （7 层）：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。
TCP/IP 分层（4 层）：网络接口层、 网际层、运输层、 应用层。
五层协议 （5 层）：物理层、数据链路层、网络层、运输层、 应用层。
每一层的协议如下：
物理层：RJ45、CLOCK、IEEE802.3 （中继器，集线器，网关）
数据链路：PPP、FR、HDLC、VLAN、MAC （网桥，交换机）

网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP、 （路由器）
传输层：TCP、UDP、SPX
会话层：NFS、SQL、NETBIOS、RPC
表示层：JPEG、MPEG、ASII
应用层：FTP、DNS、Telnet、SMTP、HTTP、WWW、NFS
每一层的作用如下：
物理层：通过媒介传输比特,确定机械及电气规范（比特 Bit）
数据链路层：将比特组装成帧和点到点的传递（帧 Frame）
网络层：负责数据包从源到宿的传递和网际互连（包 PackeT）
传输层：提供端到端的可靠报文传递和错误恢复（段 Segment）
会话层：建立、管理和终止会话（会话协议数据单元 SPDU）
表示层：对数据进行翻译、加密和压缩（表示协议数据单元 PPDU）
应用层：允许访问 OSI 环境的手段（应用协议数据单元 APDU）

31.HTTP 的长连接和短连接?

HTTP 的长连接和短连接本质上是 TCP 长连接和短连接。HTTP 属于应用层协议.
短连接:浏览器和服务器每进行一次 HTTP 操作，就建立一次连接，但任务结束就中断连接。
长连接:当一个网页打开完成后,客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。Keep-Alive 不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如 Apache）中设定这个时间。实现长连接要客户端和服务端都支持长连接。
TCP 短连接: client 向 server 发起连接请求，server 接到请求，然后双方建立连接。client 向 server 发送消息，server 回应 client，然后一次读写就完成了，这时候双方任何一个都可以发起 close 操作，不过一般都是 client 先发起 close 操作.短连接一般只会在 client/server 间传递一次读写操作
TCP 长连接: client 向 server 发起连接，server 接受 client 连接，双方建立连接。Client 与 server 完成一次读写之后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。

32.运输层协议与网络层协议的区别?

网络层协议负责的是提供主机间的逻辑通信
运输层协议负责的是提供进程间的逻辑通信

33.数据链路层协议可能提供的服务？

成帧、链路访问、透明传输、可靠交付、流量控制、差错检测、差错纠正、半双工和全双工。最重要的是帧定界（成帧）、透明传输以及差错检测。
34.IP 地址的分类
A 类地址：以 0 开头， 第一个字节范围：0~127（1.0.0.1－126.255.255.254）；
B 类地址：以 10 开头， 第一个字节范围：128~191（128.0.0.1 －191.255.255.254）；
C 类地址：以 110 开头， 第一个字节范围：192~223（192.0.0.1-223.255.255.254）；
10.0.0.0—10.255.255.255， 172.16.0.0—172.31.255.255， 192.168.0.0—192.168.255.255。（Internet 上保留地址用于内部）
IP 地址与子网掩码相与得到主机号

34. 为什么 TCP 连接要建立三次连接？

为了防止失效的连接请求又传送到主机，因而产生错误。
如果使用的是两次握手建立连接，假设有这样一种场景，客户端发送了第一个请求连接并且没有丢失，只是因为在网络结点中滞留的时间太长了，由于 TCP 的客户端迟迟没有收到确认报文，以为服务器没有收到，此时重新向服务器发送这条报文，此后客户端和服务器经过两次握手完成连接，传输数据，然后关闭连接。此时此前滞留的那一次请求连接，网络通畅了到达了服务器，这个报文本该是失效的，但是，两次握手的机制将会让客户端和服务器再次建立连接，这将导致不必要的错误和资源的浪费。
如果采用的是三次握手，就算是那一次失效的报文传送过来了，服务端接受到了那条失效报文并且回复了确认报文，但是客户端不会再次发出确认。由于服务器收不到确认，就知道客户端并没有请求连接。

35.为什么要 4 次挥手？

TCP 协议是一种面向连接的、可靠的、基于字节流的传输层通信协议，是一个全双工模式：
1、当主机 A 确认发送完数据且知道 B 已经接受完了，想要关闭发送数据口（当然确认信号还是可以发），就会发 FIN 给主机 B。
2、主机 B 收到 A 发送的 FIN，表示收到了，就会发送 ACK 回复。
3、但这是 B 可能还在发送数据，没有想要关闭数据口的意思，所以 FIN 与 ACK 不是同时发送的，而是等到 B 数据发送完了，才会发送 FIN 给主机 A。
4、A 收到 B 发来的 FIN，知道 B 的数据也发送完了，回复 ACK， A 等待 2MSL 以

后，没有收到 B 传来的任何消息，知道 B 已经收到自己的 ACK 了，A 就关闭链接，B 也关闭链接了。
确保数据能够完成传输。

36.如果已经建立了连接，但是客户端突然出现故障了怎么办？

TCP 还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为 2 小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔 75 分钟发送一次。若一连发送 10 个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

37.TCP 和 UDP 的区别？

区别	TCP	UDP
1.连接	面向连接	面向非连接
2.可靠性	可靠	非可靠
3.有序性	有序	不保证有序
4.速度	慢	快
5.量级	重量级	轻量级
6.拥塞控制或流量控制	有	没有
7	面向字节流，无记录边界	面向报文，有记录边界
8	只能单播	可以广播或组播
9.应用场景	效率低，准确性高	效率高，准确性低

38.TCP 对应的协议

（1） FTP：定义了文件传输协议，使用 21 端口。
（2） Telnet：一种用于远程登陆的端口，使用 23 端口，用户可以以自己的身份远程连接到计算机上，可提供基于 DOS 模式下的通信服务。
（3） SMTP：邮件传送协议，用于发送邮件。服务器开放的是 25 号端口。
（4） POP3：它是和 SMTP 对应，POP3 用于接收邮件。POP3 协议所用的是 110 端口。
（5）HTTP：是从 Web 服务器传输超文本到本地浏览器的传送协议

39.UDP 对应的协议

（1） DNS：用于域名解析服务，将域名地址转换为 IP 地址。DNS 用的是 53 号端口。
（2） SNMP：简单网络管理协议，使用 161 号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。
（3） TFTP(Trival File Transfer Protocal)，简单文件传输协议，该协议在熟知端口 69 上使用 UDP 服务。

40.端口及对应的服务？

服务	端口号	服务	端口号
FTP	21	SSH	22
telnet	23	SMTP	25
Domain(域名服务器)	53	HTTP	80
POP3	110	NTP（网络时间协议）	123
MySQL数据库服务	3306	Shell或 cmd	514
POP-2	109	SQL Server	1433

41.TCP/IP 的流量控制？

利用滑动窗口实现流量控制，如果发送方把数据发送得过快，接收方可能会来不及接收，这就会造成数据的丢失。所谓流量控制就是让发送方的发送速率不要太快，要让接收方来得及接收。
TCP 为每一个连接设有一个持续计时器(persistence timer)。只要 TCP 连接的一方收到对方的零窗口通知，就启动持续计时器。若持续计时器设置的时间到期，就发送一个零窗口控测报文段（携 1 字节的数据），那么收到这个报文段的一方就重新设置持续计时器。

42.TCP 拥塞控制？

防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。拥塞控制所要做的都有一个前提：网络能够承受现有的网络负荷。拥塞控制是一个全局性的过程，涉及到所有的主机、路由器，以及与降低网络传输性能有关的所有因素。

拥塞控制代价：需要获得网络内部流量分布的信息。在实施拥塞控制之前，还需要在结点之间交换信息和各种命令，以便选择控制的策略和实施控制。这样就产生了额外的开销。拥塞控制还需要将一些资源分配给各个用户单独使用，使得网络资源不能更好地实现共享。

43. HTTP1 与 HTTP2 的主要区别

Web 性能的终极目标是减少到用户端的延迟，让用户能够尽快的打开前端网页并进行相关交互。尽可能发送少的数据给服务器，从服务端下载尽可能少的数据，尽可能减少往返 （Round Trips） ，客户端与服务器无论是哪一边，额外的数据流都会带来额外的延迟开销，与此同时也更容易出现拥塞和丢包问题,这无疑严重影响了性能。 多余的 Round Trip 同样会增加延迟，尤其是在移动网络下（100ms 是让用户感觉到系统立即做出响应的时间上限）。
HTTP/2 试图解决 HTTP/1.1 的许多缺点和不灵活之处
HTTP1.x
缺陷：
线程阻塞，在同一时间，同一域名的请求有一定数量限制，超过限制数目的请求会被阻塞。
HTTP1.0
缺陷：
浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个 TCP 连接（TCP 连接的新建成本很高，因为需要客户端和服务器三次握手），服务器完成请求处理后立即断开 TCP 连接，服务器不跟踪每个客户也不记录过去的请求；
解决方案：
添加头信息——非标准的 Connection 字段 Connection: keep-alive
HTTP1.1
改进点：
1、持久连接
引入了持久连接，即 TCP 连接默认不关闭，可以被多个请求复用，不用声明 Connection: keep-alive(对于同一个域名，大多数浏览器允许同时建立 6 个持久连接)
2、管道机制
即在同一个 TCP 连接里面，客户端可以同时发送多个请求。
3、分块传输编码
即服务端没产生一块数据，就发送一块，采用”流模式”而取代”缓存模式”。
4、新增请求方式
PUT:请求服务器存储一个资源;
DELETE：请求服务器删除标识的资源；

OPTIONS：请求查询服务器的性能，或者查询与资源相关的选项和需求；
TRACE：请求服务器回送收到的请求信息，主要用于测试或诊断；
CONNECT：保留将来使用
缺点：
虽然允许复用 TCP 连接，但是同一个 TCP 连接里面，所有的数据通信是按次序进行的。服务器只有处理完一个请求，才会接着处理下一个请求。如果前面的处理特别慢，后面就会有许多请求排队等着。这将导致“队头堵塞”
避免方式：
一是减少请求数（代码合并、图片精灵），二是同时多开持久连接（静态资源分布到不同的域下）。
HTTP/2.0：
特点
采用二进制格式而非文本格式；完全多路复用，而非有序并阻塞的、只需一个连接即可实现并行；使用报头压缩，降低开销服务器推送
1. 二进制协议
HTTP/1.1 版的头信息肯定是文本（ASCII 编码），数据体可以是文本，也可以是二进制。HTTP/2 则是一个彻底的二进制协议，头信息和数据体都是二进制，并且统称为”帧”：头信息帧和数据帧。二进制协议解析起来更高效、“线上”更紧凑，更重要的是错误更少。
2. 完全多路复用（多路复用的单一长连接）
a.单一长链接
在 HTTP/2 中，客户端向某个域名的服务器请求页面的过程中，只会创建一条 TCP 连接，即使这页面可能包含上百个资源。而之前的 HTTP/1.x 一般会创建 6-8 条 TCP 连接来请求这 100 多个资源。单一的连接应该是 HTTP2 的主要优势，单一的连接能减少 TCP 握手带来的时延（如果是建立在 SSL/TLS 上面，HTTP2 能减少很多不必要的 SSL 握手，大家都知道 SSL 握手很慢吧）。
另外我们知道，TCP 协议有个滑动窗口，有慢启动这回事，就是说每次建立新连接后，数据先是慢慢地传，然后滑动窗口慢慢变大，才能较高速度地传，这下倒好，这条连接的滑动窗口刚刚变大，http1.x 就创个新连接传数据（这就好比人家 HTTP2 一直在高速上一直开着，你 HTTP1.x 是一辆公交车走走停停）。由于这种原因，让原本就具有突发性和短时性的 HTTP 连接变的十分低效。
所以咯，HTTP2 中用一条单一的长连接，避免了创建多个 TCP 连接带来的网络开销，提高了吞吐量。
b.多路复用
HTTP2 把要传输的信息分割成一个个二进制帧，首部信息会被封装到 HEADER Frame，相应的 request body 就放到 DATA Frame,一个帧你可以看成路上的一辆车,只要给这些车编号，让 1 号车都走 1 号门出，2 号车都走 2 号门出，就把不同的 http 请求或者响应区分开来了。但是，这里要求同一个请求或者响应的帧必须是有有序的，要保证 FIFO 的，但是不同的请求或者响应帧可以互相穿插。这就是 HTTP2 的多路复用，是不是充分利用了网络带宽，是不是提高了并发度？更进一步，http2 还能对这些流(车道)指定优先级，优先级能动态的被改变，例如把 CSS 和 JavaScript 文件设置得比图片的优先级要高，这样代码文件能更快的下载下来并得到执行。
c.报头压缩

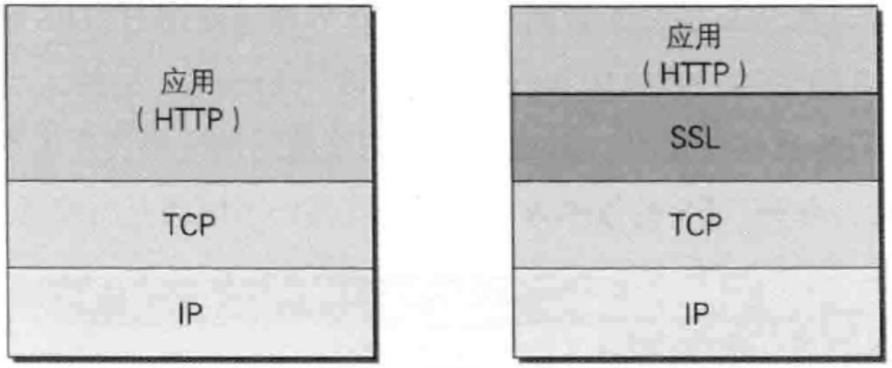
HTTP 协议是没有状态，导致每次请求都必须附上所有信息。所以，请求的很多头字段都是重复的，比如 Cookie，一样的内容每次请求都必须附带，这会浪费很多带宽，也影响速度。对于相同的头部，不必再通过请求发送，只需发送一次；HTTP/2 对这一点做了优化，引入了头信息压缩机制；一方面，头信息使用 gzip 或 compress 压缩后再发送；另一方面，客户端和服务器同时维护一张头信息表（分静态部分和动态部分），所有字段都会存入这个表，产生一个索引号，之后就不发送同样字段了，只需发送索引号。

b.服务器推送

HTTP/2 允许服务器未经请求，主动向客户端发送资源；通过推送那些服务器任务客户端将会需要的内容到客户端的缓存中，避免往返的延迟。

43.HTTPS 安全的局限性

HTTPS 因为多了一层 SSL/TLS 加密，以及数字证书的握手，数据的加密/解密，以及密钥的交换与确认，消耗更多的 CPU 和内存资源，加密范围也比较有限，黑客攻击、拒绝服务攻击、服务器劫持方面几乎起不到什么作用。最关键的，SSL 证书信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间攻击一样可行。



44.TCP 的连接管理

传输控制协议（TCP，Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议。

TCP 旨在适应支持多网络应用的分层协议层次结构。 连接到不同但互连的计算机通信网络的主计算机中的成对进程之间依靠 TCP 提供可靠的通信服务。TCP 假设它可以从较低级别的协议获得简单的，可能不可靠的数据报服务。 原则上，TCP 应该能够在从硬线连接到分组交换或电路交换网络的各种通信系统之上操作。

45.交换机与路由器有什么区别？

交换机与路由器有什么区别？
①工作所处的 OSI 层次不一样，交换机工作在 OSI 第二层数据链路层，路由器工作在 OSI 第三层网络层
②寻址方式不同：交换机根据 MAC 地址寻址，路由器根据 IP 地址寻址
③转发速不同：交换机的转发速度快，路由器转发速度相对较慢。

46.ICMP 协议？

ICMP 是 InternetControl Message Protocol，因特网控制报文协议。它是 TCP/IP 协议族的一个子协议，用于在 IP 主机、路由器之间传递控制消息。控制消息是指网络通不通、主机是否可达、路由器是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。ICMP 报文有两种：差错报告报文和询问报文。

47.DHCP 协议？

动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。通常被应用在大型的局域网络环境中，主要作用是集中的管理、分配 IP 地址，使网络环境中的主机动态的获得 IP 地址、Gateway 地址、DNS 服务器地址等信息，并能够提升地址的使用率。

48.网桥的作用？

网桥是一个局域网与另一个局域网之间建立连接的桥梁

49.数据链路层协议可能提供的服务？

成帧、链路访问、透明传输、可靠交付、流量控制、差错检测、差错纠正、半双工和全双工。最重要的是帧定界（成帧）、透明传输以及差错检测。

50.网络接口卡（网卡）的功能？

（1）进行串行/并行转换。
（2）对数据进行缓存。
（3）在计算机的操作系统安装设备驱动程序。
（4）实现以太网协议。

51.私有（保留）地址？

A 类：10.0.0.0——10.255.255.255
B 类：172.16.0.0——172.31.255.255
C 类：192.168.0.0——192.168.255.255

52.TTL 是什么？作用是什么？哪些工具会用到它（ping traceroute ifconfig　netstat）？

TTL 是指生存时间，简单来说，它表示了数据包在网络中的时间，经过一个路由器后 TTL 就减一，这样 TTL 最终会减为 0，当 TTL 为 0 时，则将数据包丢弃，这样也就是因为两个路由器之间可能形成环，如果没有 TTL 的限制，则数据包将会在这个环上一直死转，由于有了 TTL，最终 TTL 为 0 后，则将数据包丢弃。ping 发送数据包里面有 TTL，但是并非是必须的，即是没有 TTL 也是能正常工作的，traceroute 正是因为有了 TTL 才能正常工作，ifconfig 是用来配置网卡信息的，不需要 TTL，netstat 是用来显示路由表的，也是不需要 TTL 的。

53.路由表是做什么用的？在 Linux 环境中怎么配置一条默认路由？

路由表是用来决定如何将一个数据包从一个子网传送到另一个子网的，换句话说就是用来决定从一个网卡接收到的包应该送到哪一个网卡上去。路由表的每一行至少有目标网络号、子网掩码、到这个子网应该使用的网卡这三条信息。当路由器从一个网卡接收到一个包时，它扫描路由表的每一行，用里面的子网掩码与数据包中的目标 IP 地址做逻辑与运算（&）找出目标网络号。如果得出的结果网络号与这一行的网络号相同，就将这条路由表六下来作为备用路由。如果已经有备用路由了，就载这两条路由里将网络号最长的留下来，另一条丢掉（这是用无分类编址 CIDR 的情况才是匹配网络号最长的，其他的情况是找到第一条匹配的行时就可以进行转发了）。如此接着扫描下一行直到结束。如果扫描结束仍没有找到任何路由，就用默认路由。确定路由后，直接将数据包送到对应的网卡上去。在具体的实现中，路由表可能包含更多的信息为选路由算法的细节所用。

在 Linux 上可以用“route add default gw<默认路由器 IP>”命令配置一条默认路由。

54.RARP？

逆地址解析协议，作用是完成硬件地址到 IP 地址的映射，主要用于无盘工作站，因为给无盘工作站配置的 IP 地址不能保存。

55.TCP 特点

TCP 是面向连接的
通信前需要建立连接，通信结束需要释放连接。
TCP 提供可靠交付服务
所谓『可靠』指的是：TCP 发送的数据无重复、无丢失、无错误、与发送端顺序一致。
TCP 是面向字节流的
所谓『面向字节流』指的是：TCP 以字节为单位。虽然传输的过程中数据被划分成一个个数据报，但这只是为了方便传输，接收端最终接受到的数据将与发送端的数据一模一样。
TCP 提供全双工通信
所谓『全双工通信』指的是：TCP 的两端既可以作为发送端，也可以作为接收端。
一条 TCP 连接的两端只能有两个端点
TCP 只能提供点到点的通信，而 UDP 可以任意方式的通信

56.TCP 连接 与 套接字

什么是『TCP 连接』？
TCP 连接是一种抽象的概念，表示一条可以通信的链路。每条 TCP 连接有且仅有两个端点，表示通信的双方。且双发在任意时刻都可以作为发送者和接收者。
什么是『套接字』？
一条 TCP 连接的两端就是两个套接字。套接字=IP 地址: 端口号。因此，TCP 连接=（套接字 1，套接字 2）=（IP1:端口号 1，IP2:端口号 2）

57.TCP 可靠传输的实现

TCP 的可靠性表现在：它向应用层提供的数据是 无差错的、有序的、无丢失的，

简单的说就是：TCP 最终递交给应用层的数据和发送者发送的数据是一模一样的。TCP 采用了流量控制、拥塞控制、连续 ARQ 等技术来保证它的可靠性。

PS：网络层传输的数据单元为『数据报』，传输层的数据单元为『报文段』，但为了方便起见，可以统称为『分组』。

58.AQR 协议

ARQ(Automatic Repeat reQuest)自动重传请求。顾名思义，当请求失败时它会自动重传，直到请求被正确接收为止。这种机制保证了每个分组都能被正确接收。停止等待协议是一种 ARQ 协议。

59.停止等待协议的原理

无差错的情况
A 向 B 每发送一个分组，都要停止发送，等待 B 的确认应答；A 只有收到了 B 的确认应答后才能发送下一个分组。
分组丢失和出现差错的情况
发送者拥有超时计时器。每发送一个分组便会启动超时计时器，等待 B 的应答。若超时仍未收到应答，则 A 会重发刚才的分组。
分组出现差错：若 B 收到分组，但通过检查和字段发现分组在运输途中出现差错，它会直接丢弃该分组，并且不会有任何其他动作。A 超时后便会重新发送该分组，直到 B 正确接收为止。
分组丢失：若分组在途中丢失，B 并没有收到分组，因此也不会有任何响应。当 A 超时后也会重传分组，直到正确接收该分组的应答为止。
综上所述：当分组丢失 或 出现差错 的情况下，A 都会超时重传分组。
应答丢失 和 应答迟到 的情况
TCP 会给每个字节都打上序号，用于判断该分组是否已经接收。
应答丢失：若 B 正确收到分组，并已经返回应答，但应答在返回途中丢失了。此时 A 也收不到应答， 从而超时重传。紧接着 B 又收到了该分组。接收者根据序号来判断当前收到的分组是否已经接收，若已接收则直接丢弃，并补上一个确认应答。
应答迟到：若由于网络拥塞，A 迟迟收不到 B 发送的应答，因此会超时重传。B 收到该分组后，发现已经接收，便丢弃该分组，并向 A 补上确认应答。A 收到应答后便继续发送下一个分组。但经过了很长时间后，那个失效的应答最终抵达了 A，此时 A 可根据序号判断该分组已经接收，此时只需简单丢弃即可。

60.停止等待协议的注意点

每发送完一个分组，该分组必须被保留，直到收到确认应答为止。
必须给每个分组进行编号。以便按序接收，并判断该分组是否已被接收。
必须设置超时计时器。每发送一个分组就要启动计时器，超时就要重发分组。
计时器的超时时间要大于应答的平均返回时间，否则会出现很多不必要的重传，降低传输效率。但超时时间也不能太长。

61.流量控制的目的？

流量控制根本目的是防止分组丢失，它是构成 TCP 可靠性的一方面。

62.如何实现流量控制？

由滑动窗口协议（连续 ARQ 协议）实现。滑动窗口协议既保证了分组无差错、有序接收，也实现了流量控制。

63.流量控制引发的死锁

当发送者收到了一个窗口为 0 的应答，发送者便停止发送，等待接收者的下一个应答。但是如果这个窗口不为 0 的应答在传输过程丢失，发送者一直等待下去，而接收者以为发送者已经收到该应答，等待接收新数据，这样双方就相互等待，从而产生死锁。

64.持续计时器

为了避免流量控制引发的死锁，TCP 使用了持续计时器。每当发送者收到一个零窗口的应答后就启动该计时器。时间一到便主动发送报文询问接收者的窗口大小。若接收者仍然返回零窗口，则重置该计时器继续等待；若窗口不为 0，则表示应答报文丢失了，此时重置发送窗口后开始发送，这样就避免了死锁的产生。

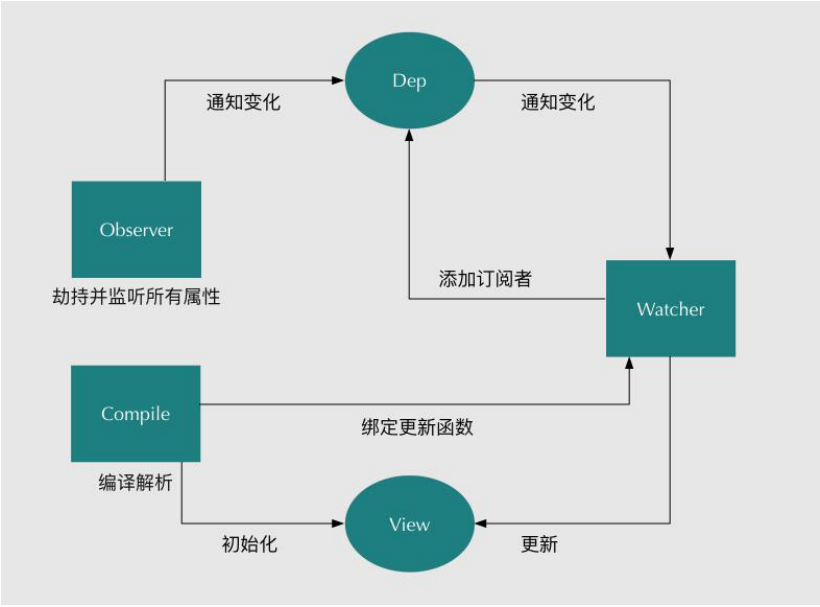
前端面试题集锦——Vue

1.vue.js 的两个核心是什么？

数据驱动和组件化。

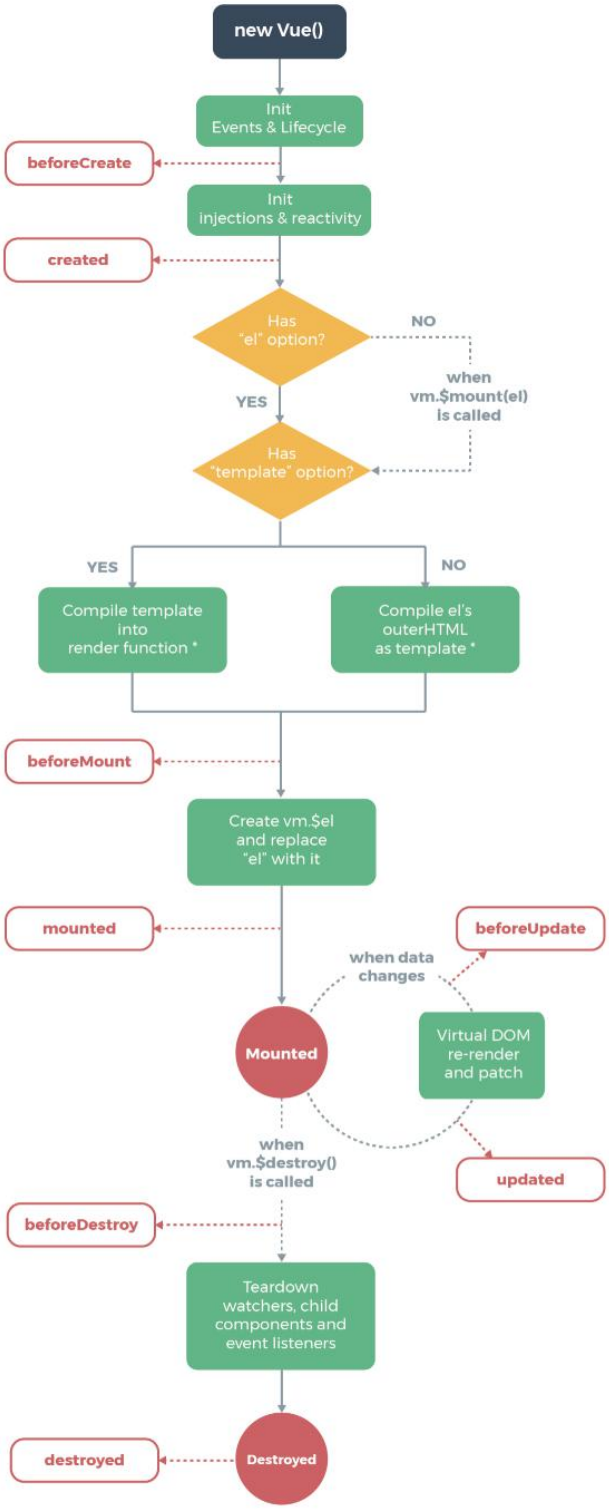
2.vue 的双向绑定的原理是什么？

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。具体实现过程：我们已经知道实现数据的双向绑定，首先要对数据进行劫持监听，所以我们需要设置一个监听器 Observer，用来监听所有属性。如果属性发上变化了，就需要告诉订阅者 Watcher 看是否需要更新。因为订阅者是有很多个，所以我们需要有一个消息订阅器 Dep 来专门收集这些订阅者，然后在监听器 Observer 和订阅者 Watcher 之间进行统一管理的。接着，我们还需要有一个指令解析器 Compile，对每个节点元素进行扫描和解析，将相关指令对应初始化成一个订阅者 Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者 Watcher 接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。因此接下去我们执行以下 3 个步骤，实现数据的双向绑定：
1.实现一个监听器 Observer，用来劫持并监听所有属性，如果有变动的，就通知订阅者。
2.实现一个订阅者 Watcher，可以收到属性的变化通知并执行相应的函数，从而更新视图。
3.实现一个解析器 Compile，可以扫描和解析每个节点的相关指令，并根据初始化模板数据以及初始化相应的订阅器。
流程图如下：



3.vue 生命周期钩子函数有哪些？

总共分为 8 个阶段创建前/后，载入前/后，更新前/后，销毁前/后。具体执行流程查看下图。



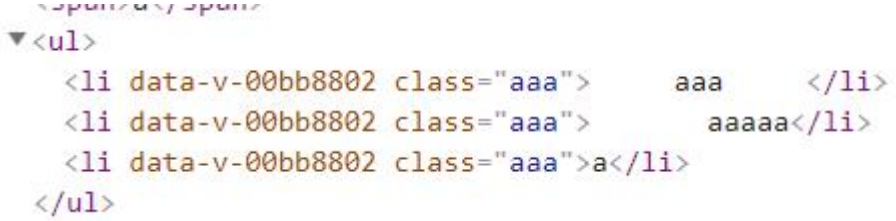
4.请问 v-if 和 v-show 有什么区别？

相同点： 两者都是在判断 DOM 节点是否要显示。
不同点：
a.实现方式： v-if 是根据后面数据的真假值判断直接从 Dom 树上删除或重建元素节点。v-show 只是在修改元素的 css 样式，也就是 display 的属性值，元素始终在 Dom 树上。
b.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；
c.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； 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 元素始终被保留；
d.性能消耗：v-if 有更高的切换消耗，不适合做频繁的切换； v-show 有更高的初始渲染消耗，适合做频繁的额切换。

5.vue 常用的修饰符

a、按键修饰符
如：.delete（捕获“删除”和”退格“键） 用法上和事件修饰符一样，挂载在 v-on: 后面，语法： v-on:keyup.xxx= ’ yyy ’ <inputclass = 'aaa' v-model="inputValue" @keyup.delete="onKey"/>
b、系统修饰符
可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器
.ctrl
.alt
.shift
.meta
c、鼠标按钮修饰符
.left
.right
.middle
这些修饰符会限制处理函数仅响应特定的鼠标按钮。如：<button @click.middle ="onClick">A</button> 鼠标滚轮单击触发 Click 默认是鼠标左键单击
d、其他修饰符
.lazy
在默认情况下，v-model 在每次 input 事件触发后将输入框的值与数据进行同步 ，我们可以添加 lazy 修饰符，从而转变为使用 change 事件进行同步：
<inputv-model.lazy="msg" >
.number
如果想自动将用户的输入值转为数值类型，可以给 v-model 添加 .number 修饰符：
<input v-model.number="age" type="number">

这通常很有用，因为即使在 type="number"时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 parseFloat()解析，则会返回原始的值。
.trim
如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符：
<inputv-model.trim="msg">



同样前面都有空格加上.trim 后 将前后空格都去掉了。

6.nextTick

在下次 dom 更新循环结束之后执行延迟回调，可用于获取更新后的 dom 状态
新版本中默认是 microtasks, v-on 中会使用 macrotasks
macrotasks 任务的实现:
osetImmediate / MessageChannel / setTimeout

7.什么是 vue 生命周期

Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、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。

8.数据响应(数据劫持)

看完生命周期后，里面的 watcher 等内容其实是数据响应中的一部分。数据响应的实现由两部分构成: 观察者(watcher) 和 依赖收集器(Dep)，其核心是 defineProperty 这个方法，它可以 重写属性的 get 与 set 方法，从而完成监听数据的改变。
Observe (观察者)观察 props 与 state
o 遍历 props 与 state，对每个属性创建独立的监听器(watcher)
使用 defineProperty 重写每个属性的 get/set(defineReactive）
oget: 收集依赖
Dep.depend()
watcher.addDep()
oset: 派发更新

Dep.notify()
watcher.update()
queenWatcher()
nextTick
flushScheduleQueue
watcher.run()
updateComponent()

大家可以先看下面的数据相应的代码实现后，理解后就比较容易看懂上面的简单脉络了。

```
let data = {a: 1}// 数据响应性
observe(data)
// 初始化观察者 new Watcher(data, 'name', updateComponent)
data.a = 2
// 简单表示用于数据更新后的操作 function updateComponent() {
    vm._update() // patchs
}
// 监视对象 function observe(obj) {
    // 遍历对象，使用 get/set 重新定义对象的每个属性值
    Object.keys(obj).map(key => {
        defineReactive(obj, key, obj[key])
    })
}
function defineReactive(obj, k, v) {
    // 递归子属性
    if (type(v) == 'object') observe(v)

    // 新建依赖收集器
    let dep = new Dep()
    // 定义 get/set
    Object.defineProperty(obj, k, {
        enumerable: true,
        configurable: true,
        get: function reactiveGetter() {
            // 当有获取该属性时，证明依赖于该对象，因此被添加进收集器中
            if (Dep.target) {
                dep.addSub(Dep.target)
            }
            return v
        },
        // 重新设置值时，触发收集器的通知机制
        set: function reactiveSetter(nV) {
            v = nV
            dep.nofify()
        },
    })
```

```
}
// 依赖收集器 class Dep {
    constructor() {
        this.subs = []
    }
    addSub(sub) {
        this.subs.push(sub)
    }
    notify() {
        this.subs.map(sub => {
            sub.update()
        })
    }
}

Dep.target = null
// 观察者 class Watcher {
    constructor(obj, key, cb) {
        Dep.target = this
        this.cb = cb
        this.obj = obj
        this.key = key
        this.value = obj[key]
        Dep.target = null
    }
    addDep(Dep) {
        Dep.addSub(this)
    }
    update() {
        this.value = this.obj[this.key]
        this.cb(this.value)
    }
    before() {
        callHook('beforeUpdate')
    }
}
```

9.virtual dom 原理实现

创建 dom 树
树的 diff，同层对比，输出 patchs(listDiff/diffChildren/diffProps)
o 没有新的节点，返回
o 新的节点 tagName 与 key 不变， 对比 props，继续递归遍历子树

对比属性(对比新旧属性列表):
旧属性是否存在与新属性列表中
都存在的是否有变化
是否出现旧列表中没有的新属性
otagName 和 key 值变化了，则直接替换成新节点
渲染差异
o 遍历 patchs， 把需要更改的节点取出来
o 局部更新 dom

```
// diff 算法的实现 function diff(oldTree, newTree) {
    // 差异收集
    let pathchs = {}
    dfs(oldTree, newTree, 0, pathchs)
    return pathchs
}
function dfs(oldNode, newNode, index, pathchs) {
    let curPathchs = []
    if (newNode) {
        // 当新旧节点的 tagName 和 key 值完全一致时
        if (oldNode.tagName === newNode.tagName && oldNode.key === newNode.key) {
            // 继续比对属性差异
            let props = diffProps(oldNode.props, newNode.props)
            curPathchs.push({ type: 'changeProps', props })
            // 递归进入下一层级的比较
            diffChildrens(oldNode.children, newNode.children, index, pathchs)
        } else {
            // 当 tagName 或者 key 修改了后，表示已经是全新节点，无需再比
            curPathchs.push({ type: 'replaceNode', node: newNode })
        }
    }

    // 构建出整颗差异树
    if (curPathchs.length) {
            if(pathchs[index]){
                pathchs[index] = pathchs[index].concat(curPathchs)
            } else {
                pathchs[index] = curPathchs
            }
    }
}
// 属性对比实现 function diffProps(oldProps, newProps) {
    let propsPathchs = []
    // 遍历新旧属性列表
    // 查找删除项
    // 查找修改项
```

```
    // 查找新增项
    forin(olaProps, (k, v) => {
        if (!newProps.hasOwnProperty(k)) {
            propsPathchs.push({ type: 'remove', prop: k })
        } else {
            if (v !== newProps[k]) {
                propsPathchs.push({ type: 'change', prop: k , value: newProps[k] })
            }
        }
    })
    forin(newProps, (k, v) => {
        if (!oldProps.hasOwnProperty(k)) {
            propsPathchs.push({ type: 'add', prop: k, value: v })
        }
    })
    return propsPathchs
}
// 对比子级差异 function diffChildrens(oldChild, newChild, index, pathchs) {
        // 标记子级的删除/新增/移动
    let { change, list } = diffList(oldChild, newChild, index, pathchs)
    if (change.length) {
        if (pathchs[index]) {
            pathchs[index] = pathchs[index].concat(change)
        } else {
            pathchs[index] = change
        }
    }

     // 根据 key 获取原本匹配的节点，进一步递归从头开始对比
    oldChild.map((item, i) => {
        let keyIndex = list.indexOf(item.key)
        if (keyIndex) {
            let node = newChild[keyIndex]
            // 进一步递归对比
            dfs(item, node, index, pathchs)
        }
    })
}
// 列表对比，主要也是根据 key 值查找匹配项// 对比出新旧列表的新增/删除/移动
function diffList(oldList, newList, index, pathchs) {
    let change = []
    let list = []
    const newKeys = getKey(newList)
    oldList.map(v => {
```

```
            if (newKeys.indexOf(v.key) > -1) {
                list.push(v.key)
            } else {
                list.push(null)
            }
        })

        // 标记删除
        for (let i = list.length - 1; i>= 0; i--) {
            if (!list[i]) {
                list.splice(i, 1)
                change.push({ type: 'remove', index: i })
            }
        }

        // 标记新增和移动
        newList.map((item, i) => {
            const key = item.key
            const index = list.indexOf(key)
            if (index === -1 || key == null) {
                // 新增
                change.push({ type: 'add', node: item, index: i })
                list.splice(i, 0, key)
            } else {
                // 移动
                if (index !== i) {
                    change.push({
                        type: 'move',
                        form: index,
                        to: i,
                    })
                    move(list, index, i)
                }
            }
        })

        return { change, list }
}
```

10.Proxy 相比于 defineProperty 的优势

数组变化也能监听到

不需要深度遍历监听

```
let data = { a: 1 }let reactiveData = new Proxy(data, {
    get: function(target, name){
        // ...
    },
    // ...
})
```

6. vue-router

mode

ohash

ohistory

跳转

othis.$router.push()

o<router-link to=""></router-link>

占位

o<router-view></router-view>

11.vuex

state: 状态中心
mutations: 更改状态
actions: 异步更改状态
getters: 获取状态
modules: 将 state 分成多个 modules，便于管理

12.vue 中 key 值的作用

使用 key 来给每个节点做一个唯一标识
key 的作用主要是为了高效的更新虚拟 DOM。另外 vue 中在使用相同标签名元素的过渡切换时，也会使用到 key 属性，其目的也是为了让 vue 可以区分它们，
否则 vue 只会替换其内部属性而不会触发过渡效果。

13.Vue 组件中 data 为什么必须是函数？

在 new Vue() 中，data 是可以作为一个对象进行操作的，然而在 component 中，data 只能以函数的形式存在，不能直接将对象赋值给它。
当 data 选项是一个函数的时候，每个实例可以维护一份被返回对象的独立的拷贝，这样各

个实例中的 data 不会相互影响，是独立的。

14.v-for 与 v-if 的优先级

v-for 的优先级比 v-if 高。

15.说出至少 4 种 vue 当中的指令和它的用法

v-if(判断是否隐藏)
v-for(把数据遍历出来)
v-bind(绑定属性)
v-model(实现双向绑定)

16.vue 中子组件调用父组件的方法

第一种方法是直接在子组件中通过 this.$parent.event 来调用父组件的方法。
第二种方法是在子组件里用$emit 向父组件触发一个事件，父组件监听这个事件就行了。
第三种是父组件把方法传入子组件中，在子组件里直接调用这个方法。

17.vue 中父组件调用子组件的方法

父组件利用 ref 属性操作子组件方法。
父：

```
<child ref="childMethod"></child>
```

子：

```
method: {
   test() {
        alert(1)
   }
}
```

在父组件里调用 test 即 this.$refs.childMethod.test()

18.vue 页面级组件之间传值

1.使用 vue-router 通过跳转链接带参数传参。
2.使用本地缓存 localStorge。
3.使用 vuex 数据管理传值。

19.说说 vue 的动态组件。

多个组件通过同一个挂载点进行组件的切换，is 的值是哪个组件的名称，那么页面就会显示哪个组件。
主要考查面试这 component 的 is 属性。

20.keep-alive 内置组件的作用

可以让当前组件或者路由不经历创建和销毁，而是进行缓存，凡是被 keep-alive 组件包裹的组件，除了第一次以外。不会经历创建和销毁阶段的。第一次创建后就会缓存到缓存当中。

21.递归组件的用法

组件是可以在它们自己的模板中调用自身的。不过它们只能通过 name 选项来做这件事。首先我们要知道，既然是递归组件，那么一定要有一个结束的条件，否则就会使用组件循环引用，最终出现“max stack size exceeded”的错误，也就是栈溢出。那么，我们可以使用 v-if="false"作为递归组件的结束条件。当遇到 v-if 为 false 时，组件将不会再进行渲染。

22.怎么定义 vue-router 的动态路由？怎么获取传过来的值？

动态路由的创建，主要是使用 path 属性过程中，使用动态路径参数，以冒号开头，如下：

```
{
   path: '/details/:id'
name: 'Details'
```

components: Details
}
访问 details 目录下的所有文件，如果 details/a，details/b 等，都会映射到 Details 组件上。
当匹配到/details 下的路由时，参数值会被设置到 this.$route.params 下，所以通过这个属性可以获取动态参数
this.$route.params.id

23.vue-router 有哪几种路由守卫？

路由守卫为：
全局守卫：beforeEach
后置守卫：afterEach
全局解析守卫：beforeResolve
路由独享守卫：beforeEnter

24.$route 和 $router 的区别是什么？

$router 为 VueRouter 的实例，是一个全局路由对象，包含了路由跳转的方法、钩子函数等。
$route 是路由信息对象||跳转的路由对象，每一个路由都会有一个 route 对象，是一个局部对象，包含 path,params,hash,query,fullPath,matched,name 等路由信息参数。

25. vue-router 响应路由参数的变化

```
// 监听当前路由发生变化的时候执行
watch: {
  $route(to, from){
    console.log(to.path)
    // 对路由变化做出响应
  }
}
```

(1)用 watch 检测
(2)组件内导航钩子函数

```
beforeRouteUpdate(to, from, next){
  // to do somethings
}
```

26. vue-router 传参

(1)使用 Params:
只能使用 name，不能使用 path
参数不会显示在路径上
浏览器强制刷新参数会被清空

```
// 传递参数
this.$router.push({
  name: Home,
  params: {
      number: 1 ,
      code: '999'
  }
})
// 接收参数
const p = this.$route.params
```

（2）使用 Query：
参数会显示在路径上，刷新不会被清空
name 可以使用 path 路径

```
// 传递参数
this.$router.push({
  name: Home,
  query: {
  number: 1 ,
  code: '999'
}
                })
// 接收参数
const q = this.$route.query
```

27.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。

28.vuex 有哪几种属性？

有五种，分别是 State、 Getter、Mutation 、Action、 Module。

29.vuex 的 State 特性是？

一、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于与一般 Vue 对象里面的 data
二、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新
三、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中

30. vuex 的 Getter 特性是？

一、getters 可以对 State 进行计算操作，它就是 Store 的计算属性
二、 虽然在组件内也可以做计算属性，但是 getters 可以在多组件之间复用
三、 如果一个状态只在一个组件内使用，是可以不用 getters

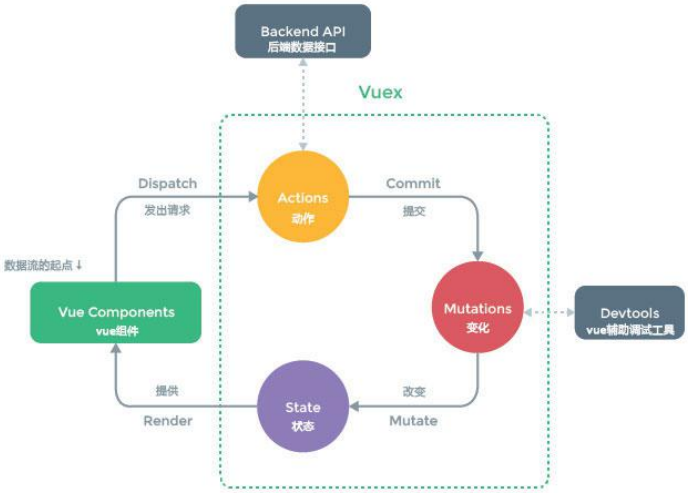
31.vuex 的 Mutation 特性是？

一、Action 类似于 mutation，不同在于：
二、Action 提交的是 mutation，而不是直接变更状态。
三、Action 可以包含任意异步操作

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

一、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。

二、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，并包装成 promise 返回，在调用处用 async await 处理返回的数据。如果不要复用这个请求，那么直接写在 vue 文件里很方便。



33.什么是 MVVM？

MVVM 是 Model-View-ViewModel 的缩写。MVVM 是一种设计思想。Model 层代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑；View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来，ViewModel 是一个同步 View 和 Model 的对象。
在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。
ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作 DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

34.mvvm 和 mvc 区别？它和其它框架（jquery）的区别是什么？哪些场景适合？

mvc 和 mvvm 其实区别并不大。都是一种设计思想。主要就是 mvc 中 Controller 演变成 mvvm 中的 viewModel。mvvm 主要解决了 mvc 中大量的 DOM 操作使页面渲

染性能降低，加载速度变慢，影响用户体验。
区别：vue 数据驱动，通过数据来显示视图层而不是节点操作。
场景：数据操作比较多的场景，更加便捷。

35.vue 的优点是什么？

低耦合。视图（View）可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的"View"上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。
可重用性。你可以把一些视图逻辑放在一个 ViewModel 里面，让很多 view 重用这段视图逻辑。
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。
可测试。界面素来是比较难于测试的，而现在测试可以针对 ViewModel 来写。

36.组件之间的传值？

父组件与子组件传值
父组件通过标签上面定义传值
子组件通过 props 方法接受数据
子组件向父组件传递数据
子组件通过$emit 方法传递参数

37.路由之间跳转

声明式（标签跳转） 编程式（ js 跳转）

38.vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

第一步：在 components 目录新建你的组件文件（indexPage.vue），script 一定要 export default {}
第二步：在需要用的页面（组件）中导入：import indexPage from

‘@/components/indexPage.vue’
第三步：注入到 vue 的子组件的 components 属性上面,components:{indexPage}
第四步：在 template 视图 view 中使用，
例如有 indexPage 命名，使用的时候则 index-page

39.vue 如何实现按需加载配合 webpack 设置

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。
不进行页面按需加载引入方式：import home from '../../common/home.vue'
进行页面按需加载的引入方式：

```
const home = r => require.ensure( [], () => r (require('../../common/home.vue')))
```

40.Vue 中引入组件的步骤?

1）采用 ES6 的 import … from …语法或 CommonJS 的 require()方法引入组件
2）对组件进行注册,代码如下

```
// 注册
Vue.component('my-component', {
    template: '<div>A custom component!</div>'
})
```

3）使用组件

41. 指令 v-el 的作用是什么?

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标.可以是 CSS 选择器，也可以是一个 HTMLElement 实例。

42. 在 Vue 中使用插件的步骤

采用 ES6 的 import … from …语法或 CommonJSd 的 require()方法引入插件
使用全局方法 Vue.use(plugin)使用插件,可以传入一个选项对象 Vue.use(MyPlugin, { someOption: true })

43.vue 生命周期的作用是什么

它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。

44.vue 生命周期总共有几个阶段

可以总共分为 8 个阶段：创建前/后，载入前/后,更新前/后,销毁前/销毁后

45.第一次页面加载会触发哪几个钩子

第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

46.DOM 渲染在 哪个周期中就已经完成

DOM 渲染在 mounted 中就已经完成了。

47.简单描述每个周期具体适合哪些场景

生命周期钩子的一些使用方法：
beforecreate : 可以在这加个 loading 事件，在加载实例时触发
created : 初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用
mounted : 挂载元素，获取到 DOM 节点
updated : 如果对数据统一处理，在这里写上相应函数
beforeDestroy : 可以做一个确认停止事件的确认框
nextTick : 更新数据后立即操作 dom

48.vue-loader 是什么？使用它的用途有哪些？

解析.vue 文件的一个加载器。
用途：js 可以写 es6、style 样式可以 scss 或 less、template 可以加 jade 等

49.scss 是什么？在 vue.cli 中的安装使用步骤是？有哪几大特性？

css 的预编译。
使用步骤：
第一步：先装 css-loader、node-loader、sass-loader 等加载器模块
第二步：在 build 目录找到 webpack.base.config.js，在那个 extends 属性中加一个拓展.scss
第三步：在同一个文件，配置一个 module 属性
第四步：然后在组件的 style 标签加上 lang 属性 ，例如：lang=”scss”
特性:
可以用变量，例如（$变量名称=值）;
可以用混合器，例如（）
可以嵌套

50.为什么使用 key？

当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

51.为什么避免 v-if 和 v-for 用在一起

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

52.VNode 是什么？虚拟 DOM 是什么？

Vue 在 页面上渲染的节点，及其子节点称为“虚拟节点 (Virtual Node)”，简写为“VNode”。“虚拟 DOM”是由 Vue 组件树建立起来的整个 VNode 树的称呼。

53.vue-loader 是什么？使用它的用途有哪些？

解析.vue 文件的一个加载器，跟 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

54.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;view 视图；app.vue 是一个应用主组件；main.js 是入口文件

55.vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

第一步：在 components 目录新建你的组件文件（smithButton.vue），script 一定要 export default {
第二步：在需要用的页面（组件）中导入：import smithButton from ‘…/components/smithButton.vue’
第三步：注入到 vue 的子组件的 components 属性上面,components:{smithButton}
第四步：在 template 视图 view 中使用，
问题有：smithButton 命名，使用的时候则 smith-button。

56.聊聊你对 Vue.js 的 template 编译的理解？

简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）
详情步骤：
首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用

以创建编译器的。另外 compile 还负责合并 option。
然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）

57.vue 路由跳转的几种方式

1、先绑定路由

```
const RouterModel = new Router({
  routes: [
        {
          path: '/cart',
          name: 'cartTest',
          meta: {
              keepAlive: true
          },
          components: {
              default: asyncLoader('cart/shop-cart')
          }
    }
  ]
});
```

2、不带参数路由跳转

```
<router-link :to="{name: 'cartTest'}">name 不带参数</router-link>
<router-link :to="{path: '/cart'}">path 不带参数</router-link>
```

3、 带参数路由跳转

```
<router-link :to="{name: 'cartTest', params: {id: 1}}">name 带参数 params 传参数 (类似 post),html 取参 $route.params.id script, 取参 this.$route.params.id</router-link>
<router-link :to="{name: 'cartTest', query: {id: 1}}">name 带参数,query 传参数 (类似 get,url 后面会显示参数),html 取参 $route.query.id, script 取参 this.$route.query.id</router-link>
```

4、 this.$router.push() 函数内调用

不带参数

```
@click="this.$router.push({name: 'cartTest'})"
@click="this.$router.push({path: '/cart'})"
@click="this.$router.push('/cart')"
```

带参数

```
@click="this.$router.push({name: 'cartTest', params: {id: 1}})"
@click="this.$router.push({name: 'cartTest', query: {id: 1}})"
```

说明：

query 和 params 区别

query 类似 get, 跳转之后页面 url 后面会拼接参数,类似?id=1, 非重要性的可以这样传，密码之类还是用 params 刷新页面 id 还在

params 类似 post, 跳转之后页面 url 后面不会拼接参数 ，但是刷新页面 id 会消失

58.vue-cli 创建自定义组件

1、新建一个.vue 文件 (一般 IDE 会自动生成 script export default)
2、编写 dom, 组件的 name 使用短横线分隔（component-name）方式。
3、在需要的页面 import componentName from ‘…’
4、组件注册，compontens :{ [componentName.name]: componentName}
5、在需要的页面 template 中显示

59.<keep-alive</keep-alive 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。
在 vue 2.1.0 版本之后， keep-alive 新加入了两个属性: include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于 include) 。
使用方法

```
<keep-alive include='include_components' exclude='exclude_components'>
  <component>
    <!-- 该组件是否缓存取决于 include 和 exclude 属性 -->
  </component>
</keep-alive>
```

参数解释
include - 字符串或正则表达式，只有名称匹配的组件会被缓存
exclude - 字符串或正则表达式，任何名称匹配的组件都不会被缓存
include 和 exclude 的属性允许组件有条件地缓存。二者都可以用“，”分隔字符串、正则表达式、数组。当使用正则或者是数组时，要记得使用 v-bind 。
使用示例

```
<!-- 逗号分隔字符串，只有组件 a 与 b 被缓存。 -->
<keep-alive include="a,b">
  <component></component>
</keep-alive>
<!-- 正则表达式 (需要使用 v-bind，符合匹配规则的都会被缓存) -->
<keep-alive :include="/a|b/">
  <component></component>
</keep-alive>
<!-- Array (需要使用 v-bind，被包含的都会被缓存) -->
<keep-alive :include="['a', 'b']">
  <component></component>
</keep-alive>
```

大白话: 比如有一个列表和一个详情，那么用户就会经常执行打开详情=>返回列表=>打开详情…这样的话列表和详情都是一个频率很高的页面，那么就可以对列表组件使用 <keep-alive</keep-alive 进行缓存，这样用户每次返回列表的时候，都能从缓存中快速渲染，

而不是重新渲染。

60.vue 如何实现按需加载配合 webpack 设置?

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。
不进行页面按需加载引入方式：
import home from '…/…/common/home.vue'
进行页面按需加载的引入方式：
const home = r => require.ensure([], () => r (require('…/…/common/home.vue')))

61.Vue 实现数据双向绑定的原理 Object.defineProperty()

vue 实现数据双向绑定主要是：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，触发相应监听回调。当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。
vue 的数据双向绑定 将 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 的数据变化，通过 Compile 来解析编译模板指令（vue 中是用来解析 {{msg}}），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 变更双向绑定效果。

62.Vue 的路由实现：hash 模式和 history 模式

hash 模式
在浏览器中符号“#”，#以及#后面的字符称之为 hash，用 window.location.hash 读取；
特点：hash 虽然在 URL 中，但不被包括在 HTTP 请求中；用来指导浏览器动作，对服务端安全无用，hash 不会重加载页面。
history 模式
history 采用 HTML5 的新特性；如果你使用 history 的话，改变路由的时候，后台会给你响应；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及 popState 监听浏览器历史记录变化，但是 pushState(),replaceState()不会触发该函数。

63.Vue 与 Angular 以及 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：
1.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。
2.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。 对于庞大的应用来说，这个优化差异还是比较明显的。
2.与 React 的区别
相同点：
1.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。
2.中心思想相同：一切都是组件，组件实例之间可以嵌套。
3.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。
4.都不内置 AJAX，Route 等功能到核心包，而是以插件的方式加载。 5.在组件开发中都支持 mixins 的特性。
不同点：
React 依赖 Virtual DOM,而 Vue.js 使用的是 DOM 模板。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 DOM。

64.vue 路由的钩子函数

第一种：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；
beforeRouteEnter
beforeRouteUpdate (2.2 新增)
beforeRouteLeave
第三种：单独路由独享组件。
beforeEnter
每个钩子方法接收三个参数：
to: Route: 即将要进入的目标 路由对象
from: Route: 当前导航正要离开的路由

next: Function: 一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。

65.route 和 router 的区别

route 是“路由信息对象”，包括 path，params，hash，query，fullPath，matched，name 等路由信息参数。而 router 是“路由实例”对象包括了路由的跳转方法，钩子函数等。

66.什么是 vue 的计算属性？

在模板中放入太多的逻辑会让模板过重且难以维护，在需要对数据进行复杂处理，且可能多次使用的情况下，尽量采取计算属性的方式。好处：①使得数据处理结构清晰；②依赖于数据，数据更新，处理结果自动更新；③计算属性内部 this 指向 vm 实例；④在 template 调用时，直接写计算属性名即可；⑤常用的是 getter 方法，获取数据，也可以使用 set 方法改变数据；⑥相较于 methods，不管依赖的数据变不变，methods 都会重新计算，但是依赖数据不变的时候 computed 从缓存中获取，不会重新计算。

67.vue 等单页面应用（spa）及其优缺点

优点： Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效、快速、模块友好；即第一次就将所有的东西都加载完成，因此，不会导致页面卡顿。
缺点： 不支持低版本的浏览器，最低只支持到 IE9；不利于 SEO 的优化（如果要支持 SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一些；不可以使用浏览器的导航按钮需要自行实现前进、后退。

68.vue-cli 如何新增自定义指令？

1.创建局部指令

```
var app = new Vue({
    el: '#app',
    data: {
    },
    // 创建指令(可以多个)
```

```
    directives: {
        // 指令名称
        dir1: {
            inserted(el) {
                // 指令中第一个参数是当前使用指令的 DOM
                console.log(el);
                console.log(arguments);
                // 对 DOM 进行操作
                el.style.width = '200px';
                el.style.height = '200px';
                el.style.background = '#000';
            }
        }
    }
})
2.全局指令
Vue.directive('dir2', {
    inserted(el) {
        console.log(el);
    }
})
3.指令的使用

<div id="app">
    <div v-dir1></div>
    <div v-dir2></div>
</div>
```

69.v-on 可以绑定多个方法吗？

可以

70. vue 中 key 值的作用？

避免 dom 节点复用，让每一次数据改变都重新渲染 dom 节点
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。

71.active-class 是哪个组件的属性？嵌套路由怎么定义？

vue-router 模块的 router-link 组件的属性。
嵌套路由：
嵌套路由顾名思义就是路由的多层嵌套。
重构 router/index.js 的路由配置，需要使用 children 数组来定义子路由 ，路由定义：

```
{
        path: '/me',
        name: 'Me',
        component: Me,
        children: [
          {
            path: 'collection',
            name: 'Collection',
            component: Collection
          },
          {
            path: 'trace',
            name: 'Trace',
            component: Trace
          }
        ]
    }
```

以“/”开头的嵌套路径会被当作根路径，所以子路由上不用加“/”;
在生成路由时，主路由上的 path 会被自动添加到子路由之前，所以子路由上的 path 不用在重新声明主路由上的 path 了。
在外层路由组件中，如下写法。

```
<template>
  <div class="me">
    <div class="tabs">
      <ul>
        <!--<router-link :to="{name: 'Default'}" tag="li" exact>默认内容</router-link>-->
        <router-link :to="{name: 'Collection'}" tag="li" >我的收藏</router-link>
        <router-link :to="{name: 'Trace'}" tag="li">我的足迹</router-link>
      </ul>
    </div>
    <div class="content">
      <router-view></router-view>
    </div>
  </div>
```

</template>

72.axios 是什么？怎么使用？描述使用它实现登录功能的流程？

请求后台资源的模块。npm install axios -S 装好，然后发送的是跨域，需在配置文件中 config/index.js 进行设置。后台如果是 Tp5 则定义一个资源路由。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。

73.axios+tp5 进阶中，调用 axios.post(‘api/user’)是进行的什么操作？axios.put(‘api/user/8′)呢？

跨域，添加用户操作，更新操作。

74.什么是 RESTful API？怎么使用？

是一个 api 的标准，无状态请求。请求的路由地址是固定的，如果是 tp5 则先路由配置中把资源路由配置好。标准有：.post .put .delete

75.请说下封装 vue 组件的过程？

首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：效率低、难维护、复用性等问题。
然后，使用 Vue.extend 方法创建一个组件，然后使用 Vue.component 方法注册组件。子组件需要数据，可以在 props 中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用 emit 方法。

76.watch 和 computed 区别

区别一：
watch 可以允许你没有返回值，对数据做一些处理，
但是 computed 必须要有返回值，他才能根据返回值，知道你函数依赖的是谁，当哪个数据改变的时候，触发该方法。
区别二：
computed 可以函数依赖很多值，但是 watch 只能依赖一个值。

77.vuex 有哪几种属性？

一、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于与一般 Vue 对象里面的 data
二、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新
三、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中。

78.vuex 的 Mutation 特性是？

一、Action 类似于 mutation，不同在于：
二、Action 提交的是 mutation，而不是直接变更状态。
三、Action 可以包含任意异步操作

79.mint-ui 或其他前端组件库在 vue 中怎么使用

基于 vue 的前端组件库通过 npm 安装，然后 import 样式和 js，vue.use（mintUi）全局引入。在单个组件局部引入：import {Toast} from 'mint-ui'。

80.自定义指令（v-check、v-focus）的方法有哪些？它有哪些钩子函数？还有哪些钩子函数参数？

全局定义指令：在 vue 对象的 directive 方法里面有两个参数，一个是指令名称，另外一个是函数。组件内定义指令：directives
钩子函数：bind（绑定事件触发）、inserted(节点插入的时候触发)、update（组件内相关更新）
钩子函数参数：el、binding。

80.pwa 是什么？

渐进式网页应用，PWA 应该具有一下特性：

渐进式：能确保每个用户都能打开网页响应式：PC，手机，平板，不管哪种格式，网页格式都能完美适配
2.离线应用：支持用户在没网的条件下也能打开网页，这里就需要 Service Worker 的帮助
3.APP 化：能够像 APP 一样和用户进行交互
4.常更新：一旦 Web 网页有什么改动，都能立即在用户端体现出来
5.安全：安全第一，给自己的网站加上一把绿锁–HTTPS
6.可搜索：能够被引擎搜索到
7.推送：做到在不打开网页的前提下，推送新的消息
8.可安装：能够将 Web 想 APP 一样添加到桌面
9.可跳转：只要通过一个连接就可以跳转到你的 Web 页面

81.怎么定义 vue-router 的动态路由？怎么获取传过来的动态参数？

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。

82.v-model 是什么？怎么使用？ vue 中标签怎么绑定事件？

可以实现双向绑定，指令（v-class、v-for、v-if、v-show、v-on）。vue 的 model 层的 data 属性。绑定事件：<input @click=doLog()/>

83.iframe 的优缺点？

iframe 也称作嵌入式框架，嵌入式框架和框架网页类似，它可以把一个网页的框架和内容嵌入在现有的网页中。
优点：
解决加载缓慢的第三方内容如图标和广告等的加载问题
Security sandbox
并行加载脚本
方便制作导航栏
缺点：
iframe 会阻塞主页面的 Onload 事件

即时内容为空，加载也需要时间
没有语意

84.简述一下 Sass、Less，且说明区别？

他们是动态的样式语言，是 CSS 预处理器,CSS 上的一种抽象层。他们是一种特殊的语法/语言而编译成 CSS。
变量符不一样，less 是@，而 Sass 是$;
Sass 支持条件语句，可以使用 if{}else{},for{}循环等等。而 Less 不支持;
Sass 是基于 Ruby 的，是在服务端处理的，而 Less 是需要引入 less.js 来处理 Less 代码输出 Css 到浏览器。

85.vuex 是什么？怎么使用？哪种功能场景使用它？

vue 框架中状态管理。在 main.js 引入 store，注入。新建了一个目录 store，….. 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

86.vue-router 是什么？它有哪些组件？

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

87.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。

88.你是怎么认识 vuex 的？

vuex 可以理解为一种开发模式或框架。比如 PHP 有 thinkphp，java 有 spring 等。
通过状态（数据源）集中管理驱动组件的变化（好比 spring 的 IOC 容器对 bean 进行集中管理）。
应用级的状态集中放在 store 中；改变状态的方式是提交 mutations，这是个同步的事物；异步逻辑应该封装在 action 中。

89.简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）

详情步骤：
首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用以创建编译器的。另外 compile 还负责合并 option。
然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）
vue 的历史记录
history 记录中向前或者后退多少步
vuejs 与 angularjs 以及 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：
1.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。

2.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。
对于庞大的应用来说，这个优化差异还是比较明显的。
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 依赖 Virtual DOM,而 Vue.js 使用的是 DOM 模板。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 DOM。

90. v-show 和 v-if 指令的共同点和不同点

v-show 指令是通过修改元素的 display 的 CSS 属性让其显示或者隐藏
v-if 指令是直接销毁和重建 DOM 达到让元素显示和隐藏的效果

91.如何让 CSS 只在当前组件中起作用

将当前组件的<style>修改为<style scoped>

92.<keep-alive></keep-alive>的作用是什么？

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染。

93.vue 中标签怎么绑定事件

绑定事件：<input @click="rdhub.cn" />

94.vue 与 react 的异同

相同点：
都支持组件化与虚拟 DOM
都支持 props 进行父子组件通信
都支持数据驱动视图，不支持操作 真实 DOM，更新状态视图自动更新
都支持服务器端渲染
不同点：
设计思想不同，vue 是 MVVM，react 是 MVC
vue 数据双向绑定，react 数据单向绑定
组件写法不同。react 推荐 jsx 写法，即把 html 与 css 都写进 js; vue 推荐 webpack+vue-loader 的单文件组件格式，即 html、css 与 js 都写进同一个文件
state 对象在 react 中是不可变的，需要使用 setState 方法更新状态；而 vue 中 state 不是必须的，数据由 data 在 vue 对象中管理
虚拟 DOM 不一样，vue 会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树；react 每当应用状态改变时，全部组件都会重新渲染，所以 react 需要 shouldComponentUpdate 这个生命周期方法来进行控制

95.组件 data 为什么要用函数

vue 组件中 data 值不能为对象，因为对象是引用类型，组件可能会被多个实例同时引用。如果 data 值为对象，将导致多个实例共享一个对象，其中一个组件改变 data 属性值，其它实例也会受到影响。
只有当 data 为函数时，通过 return 返回对象的拷贝，致使每个实例都有自己独立的对象，实例之间可以互不影响的改变 data 属性值。

96.ajax 和 axios、fetch 的区别？

ajax
传统 Ajax 指的是 XMLHttpRequest（XHR）， 最早出现的发送后端请求技术，隶属于原始 js 中，核心使用 XMLHttpRequest 对象，多个请求之间如果有先后关系的话，就会出现回调地狱。
juery.ajax

```
$.ajax({
    type: 'POST',
    url: url,
    data: data,
    dataType: dataType,
```

```
    success: function () {},
    error: function () {}
});
```

JQuery ajax 是对原生 XHR 的封装，除此以外还增添了对 JSONP 的支持,举出几个缺点

本身是针对 MVC 的编程,不符合现在前端 MVVM 的浪潮

基于原生的 XHR 开发，XHR 本身的架构不清晰。

JQuery 整个项目太大，单纯使用 ajax 却要引入整个 JQuery 非常的不合理（采取个性化打包的方案又不能享受 CDN 服务）

不符合关注分离（Separation of Concerns）的原则

配置和调用方式非常混乱，而且基于事件的异步模型不友好

axios

```
axios({
      method: 'post',
      url: '/user/12345',
      data: {
            firstName: 'Fred',
            lastName: 'Flintstone'
      }
})
.then(function (response) {
      console.log(response);
})
.catch(function (error) {
      console.log(error);
});
```

axios 是一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端，本质上也是对原生 XHR 的封装，只不过它是 Promise 的实现版本，符合最新的 ES 规范，它本身具有以下特征：

1.从浏览器中创建 XMLHttpRequest

2.支持 Promise API

3.客户端支持防止 CSRF

4.提供了一些并发请求的接口（重要，方便了很多的操作）

5.从 node.js 创建 http 请求

6.拦截请求和响应

7.转换请求和响应数据

8.取消请求

9.自动转换 JSON 数据

fetch

```
try {
   let response = await fetch(url);
   let data = response.json();
   console.log(data);
} catch(e) {
   console.log("Oops, error", e);
```

}
fetch 号称是 AJAX 的替代品，是在 ES6 出现的，使用了 ES6 中的 promise 对象。Fetch 是基于 promise 设计的。Fetch 的代码结构比起 ajax 简单多了，参数有点像 jQuery ajax。但是，一定记住 fetch 不是 ajax 的进一步封装，而是原生 js，没有使用 XMLHttpRequest 对象。
fetch 的优点：
1.符合关注分离，没有将输入、输出和用事件来跟踪的状态混杂在一个对象里
2.更好更方便的写法
3.语法简洁，更加语义化
基于标准 Promise 实现，支持 async/await
同构方便，使用 isomorphic-fetch
4.更加底层，提供的 API 丰富（request, response）
5.脱离了 XHR，是 ES 规范里新的实现方式

97.说下对 Virtual DOM 算法的理解

包括几个步骤：
1、用 JavaScript 对象结构表示 DOM 树的结构，然后用这个树构建一个真正的 DOM 树，插到文档当中；
2、当状态变更的时候，重新构造一棵新的对象树，然后用新的树和旧的树进行比较，记录两棵树差异；
3、把 2 所记录的差异应用到步骤 1 所构建的真正的 DOM 树上，视图就更新了。
Virtual DOM 本质上就是在 JS 和 DOM 之间做了一个缓存。可以类比 CPU 和硬盘，既然硬盘这么慢，我们就在它们之间加个缓存：既然 DOM 这么慢，我们就在它们 JS 和 DOM 之间加个缓存。CPU（JS）只操作内存（Virtual DOM），最后的时候再把变更写入硬盘（DOM）。

98.解释单向数据流和双向数据绑定

单向数据流：顾名思义，数据流是单向的。数据流动方向可以跟踪，流动单一，追查问题的时候可以更快捷。缺点就是写起来不太方便。要使 UI 发生变更就必须创建各种 action 来维护对应的 state。
双向数据绑定：数据之间是相通的，将数据变更的操作隐藏在框架内部。优点是在表单交互较多的场景下，会简化大量与业务无关的代码。缺点就是无法追踪局部状态的变化，增加了出错时 debug 的难度。

99.Vue 如何去除 URL 中的#

vue-router 默认使用 hash 模式，所以在路由加载的时候，项目中的 URL 会自带“#”。如果不想使用“#”，可以使用 vue-router 的另一种模式 history：new Router ({ mode : 'history', routes: [ ]})
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。

100. NextTick 是做什么的

$nextTick 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。

101. Vue 组件 data 为什么必须是函数

因为 JS 本身的特性带来的，如果 data 是一个对象，那么由于对象本身属于引用类型，当我们修改其中的一个属性时，会影响到所有 Vue 实例的数据。如果将 data 作为一个函数返回一个对象，那么每一个实例的 data 属性都是独立的，不会相互影响了。

102. 计算属性 computed 和事件 methods 有什么区别

我们可以将同一函数定义为一个 method 或者一个计算属性。对于最终的结果，两种方式是相同的。
不同点：
computed：计算属性是基于它们的依赖进行缓存的，只有在它的相关依赖发生改变时才会重新求值。
method：只要发生重新渲染， method 调用总会执行该函数。

103. 对比 jQuery ，Vue 有什么不同

jQuery 专注视图层，通过操作 DOM 去实现页面的一些逻辑渲染；Vue 专注于数据层，通过数据的双向绑定，最终表现在 DOM 层面，减少了 DOM 操作。
Vue 使用了组件化思想，使得项目子集职责清晰，提高了开发效率，方便重复利用，便于协同开发。

104. Vue 中怎么自定义指令

全局注册
// 注册一个全局自定义指令 v-focus Vue .directive('focus' , { // 当被绑定的元素插入到 DOM 中时…… inserted: function (el) { // 聚焦元素 el.focus() } })
局部注册
directives: { focus: { // 指令的定义 inserted: function (el) { el.focus() } } }

105. Vue 中怎么自定义过滤器

可以用全局方法 Vue.filter() 注册一个自定义过滤器，它接收两个参数：过滤器 ID 和过滤器函数。过滤器函数以值为参数，返回转换后的值。
Vue .filter('reverse' , function (value) { return value.split('').reverse().join('') })

过滤器也同样接受全局注册和局部注册。

106. 对 keep-alive 的了解

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。
<keep-alive><component><!-- 该组件将被缓存！--></component></keep-alive>
可以使用 API 提供的 props，实现组件的动态缓存。

前端面试题集锦——算法

1. 五大算法

- **贪心算法**: 局部最优解法
- **分治算法**: 分成多个小模块，与原问题性质相同
- **动态规划**: 每个状态都是过去历史的一个总结
- **回溯法**: 发现原先选择不优时，退回重新选择
- **分支限界法**

2. 基础排序算法

- 冒泡排序: 两两比较

```
function bubleSort(arr) {
    var len = arr.length;
    for (let outer = len ; outer >= 2; outer--) {
        for(let inner = 0; inner <=outer - 1; inner++) {
            if(arr[inner] > arr[inner + 1]) {
                [arr[inner],arr[inner+1]] =
[arr[inner+1],arr[inner]]
            }
        }
    }
    return arr;
}
```

- 选择排序: 遍历自身以后的元素，最小的元素跟自己调换位置

```
function selectSort(arr) {
    var len = arr.length;
    for(let i = 0 ;i < len - 1; i++) {
        for(let j = i ; j<len; j++) {
            if(arr[j] < arr[i]) {
                [arr[i],arr[j]] = [arr[j],arr[i]];
            }
        }
    }
    return arr
}
```

- 插入排序: 即将元素插入到已排序好的数组中

```
function insertSort(arr) {
    for(let i = 1; i < arr.length; i++) {  //外循环从 1 开始，默认 arr[0]
是有序段
        for(let j = i; j > 0; j--) {  //j = i,将 arr[j]依次插入有序段中
            if(arr[j] < arr[j-1]) {
                [arr[j],arr[j-1]] = [arr[j-1],arr[j]];
            } else {
                break;
            }
        }
    }
    return arr;
}
```

3. 高级排序算法

- 快速排序
    - 选择基准值(base)，原数组长度减一(基准值)，使用 splice
    - 循环原数组，小的放左边(left 数组)，大的放右边(right 数组);
    - concat(left, base, right)
    - 递归继续排序 left 与 right

```
function quickSort(arr) {
    if(arr.length <= 1) {
        return arr;  //递归出口
    }
    var left = [],
        right = [],
        current = arr.splice(0,1);
    for(let i = 0; i < arr.length; i++) {
        if(arr[i] < current) {
            left.push(arr[i])  //放在左边
        } else {
            right.push(arr[i]) //放在右边
        }
    }
    return quickSort(left).concat(current,quickSort(right));
}
```

希尔排序：不定步数的插入排序，插入排序

口诀: 插冒归基稳定，快选堆希不稳定

排序算法	平均时间复杂度	最坏时间复杂度	空间复杂度	是否稳定
冒泡排序	O(n²)	O(n²)	O(1)	是
选择排序	O(n²)	O(n²)	O(1)	不是
直接插入排序	O(n²)	O(n²)	O(1)	是
快速排序	O(nlogn)	O(n²)	O(logn)	不是
希尔排序	O(nlogn)	O(n^s)	O(1)	不是

稳定性： 同大小情况下是否可能会被交换位置，虚拟 dom 的 diff，不稳定性会导致重新渲染；

4. 递归运用(斐波那契数列)： 爬楼梯问题

初始在第一级，到第一级有 1 种方法(s(1) = 1)，到第二级也只有一种方法(s(2) = 1)，第三级(s(3) = s(1) + s(2))

```
function cStairs(n) {
    if(n === 1 || n === 2) {
        return 1;
    } else {
        return cStairs(n-1) + cStairs(n-2)
    }
}
```

5. 数据树

- 二叉树: 最多只有两个子节点
    - 完全二叉树
    - 满二叉树
        - 深度为 h, 有 n 个节点，且满足 n = 2^h - 1
- 二叉查找树: 是一种特殊的二叉树，能有效地提高查找效率
    - 小值在左，大值在右
    - 节点 n 的所有左子树值小于 n，所有右子树值大于 n

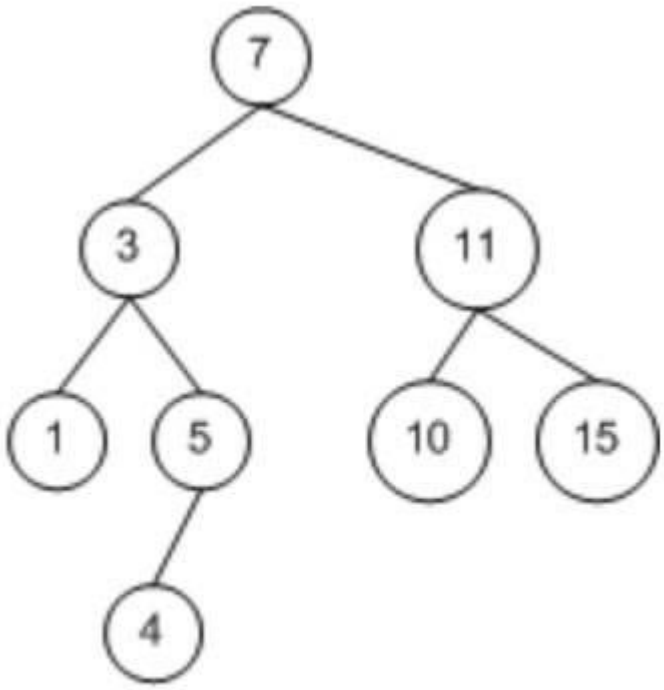


图: 二叉查找树(BST)

- 遍历节点
    - 前序遍历
        1. 根节点
        1. 访问左子节点，回到 1
        1. 访问右子节点，回到 1
    - 中序遍历
        1. 先访问到最左的子节点
        1. 访问该节点的父节点
        1. 访问该父节点的右子节点， 回到 1
    - 后序遍历
        1. 先访问到最左的子节点

1. 访问相邻的右节点

1. 访问父节点， 回到 1

- 插入与删除节点

6. 天平找次品

有 n 个硬币，其中 1 个为假币，假币重量较轻，你有一把天平，请问，至少需要称多少次能保证一定找到假币?

- 三等分算法:
  -
    1. 将硬币分成 3 组，随便取其中两组天平称量

    - 平衡，假币在未上称的一组，取其回到 1 继续循环
    - 不平衡，假币在天平上较轻的一组， 取其回到 1 继续循环

结语

由于精力时间及篇幅有限，这篇就先写到这。大家慢慢来不急。。

- Webpack 相关
  - 原理
  - Loader
  - Plugin
- 项目性能优化
  - 首屏渲染优化
  - 用户体验优化
  - webpack 性能优化
- Hybrid 与 Webview
  - webview 加载过程
  - bridge 原理
  - hybrid app 经验
- 框架: React