前端开发知识点：

HTML&CSS：

对Web标准的理解、浏览器内核差异、兼容性、hack、CSS基本功：布局、盒子模型、选择器优先级、

HTML5、CSS3、Flexbox

JavaScript：

数据类型、运算、对象、Function、继承、闭包、作用域、原型链、事件、RegExp、JSON、Ajax、DOM、BOM、内存泄漏、跨域、异步装载、模板引擎、前端MVC、路由、模块化、Canvas、ECMAScript 6、Nodejs

其他：

移动端、响应式、自动化构建、HTTP、离线存储、WEB安全、优化、重构、团队协作、可维护、易用性、SEO、UED、架构、职业生涯、快速学习能力

面试过程

1. 代码编写笔试
2. 问答试
3. 过程一般遵循 STAR 面试法 。

STAR面试法，是企业招聘面试过程中可采用的技巧。其中，“STAR”是SITUATION（背景）、TASK（任务）、ACTION（行动）和RESULT（结果）四个英文单词的首字母组合。

问题举例编辑

问题：请讲出一件你通过学习尽快胜任新的工作任务的事。追问：

（1）这件事发生在什么时候？---------------------S

（2） 你要从事的工作任务是什么？--------------T

（3） 接到任务后你怎么办？-----------------------A

（4） 你用了多长时间获得完成该任务所必须的知识？------深层次了解

（5） 你在这个过程中遇见困难了吗？ -------------------顺便了解坚韧性

（6） 你最后完成任务的情况如何？--------------R

喜欢什么样的面试者

1. 基础扎实

从多年的经验看，那些发展好的同学都具备扎实的基础知识

比如只懂 jQuery 不懂 JavaScript 是不行的哦

如果了解计算机基础会更好，因为我们将面临很多非前端技术的问题

1. 主动思考

被动完成任务的同学在这里进步会很慢

你需要有自己的想法，而不是仅仅完成任务

1. 爱学习

前端领域知识淘汰速度很快，所以最好能经常学习和接触新东西

1. 有深度

遇到问题时多研究背后深层次的原因，而不是想办法先绕过去

比如追踪某个 Bug 一直了解它本质的原因

1. 有视野

创新往往来自于不同学科的交集，如果你了解的领域越多，就越有可能有新想法

校招会更加关注基础知识，而社招会更加关注之前做过的项目情况。

HTML篇

1，Doctype作用？标准模式与兼容模式各有什么区别?

<!DOCTYPE>声明位于位于HTML文档中的第一行，处于 <html>标签之前。告知浏览器的解析器         用什么文档标准解析这个文档。DOCTYPE不存在或格式不正确会导致文档以兼容模式呈现。

标准模式的排版 和JS运作模式都是以该浏览器支持的最高标准运行。在兼容模式中，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。

2，HTML5 为什么只需要写<!DOCTYPE HTML>？

HTML5 不基于 SGML，因此不需要对DTD进行引用，但是需要doctype来规范浏览器的行为（让浏览器按照它们应该的方式来运行）；

而HTML4.01基于SGML,所以需要对DTD进行引用，才能告知浏览器文档所使用的文档类型。

3，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。

（1）行内元素有：a b span img input select strong（强调的语气）

（2）块级元素有：div ul ol li dl dt dd h1 h2 h3 h4…p

（3）常见的空元素：

<br><hr><img><input><link><meta>

鲜为人知的是：

<area><base><col><command><embed><keygen><param><source><track><wbr>

4，页面导入样式时，使用link和@import有什么区别？

link属于XHTML标签，除了加载CSS外，还能用于定义RSS, 定义rel连接属性等作用；而@import是CSS提供的，只能用于加载CSS;

页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载;

import是CSS2.1 提出的，只在IE5以上才能被识别，而link是XHTML标签，无兼容问题;

5，介绍一下你对浏览器内核的理解？

主要分成两部分：渲染引擎(layout engineer或Rendering Engine)和JS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。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。

JS引擎则：解析和执行javascript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。

6，常见的浏览器内核有哪些？

Trident内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称MSHTML]

Gecko内核：Netscape6及以上版本，FF,MozillaSuite/SeaMonkey等

Presto内核：Opera7及以上。      [Opera内核原为：Presto，现为：Blink;]

Webkit内核：Safari,Chrome等。   [ Chrome的：Blink（WebKit的分支）]

7，html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分HTML 和HTML5？

\* HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

绘画 canvas;

用于媒介回放的 video 和 audio 元素;

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;

语意化更好的内容元素，比如 article、footer、header、nav、section;

表单控件，calendar、date、time、email、url、search;

新的技术webworker, websocket, Geolocation;

移除的元素：

纯表现的元素：basefont，big，center，font, s，strike，tt，u;

对可用性产生负面影响的元素：frame，frameset，noframes；

\* 支持HTML5新标签：

IE8/IE7/IE6支持通过document.createElement方法产生的标签，

可以利用这一特性让这些浏览器支持HTML5新标签，

浏览器支持新标签后，还需要添加标签默认的样式。

当然也可以直接使用成熟的框架、比如html5shim;

                  <!--[if lt IE 9]>

                                    <script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>

                  <![endif]-->

\* 如何区分HTML5： DOCTYPE声明\新增的结构元素\功能元素

H5新特性

表单 画布 音视频 地理定位 媒体查询 css新特性 离线缓存 本地存储 拖拽

8，简述一下你对HTML语义化的理解？

用正确的标签做正确的事情。

html语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析;

即使在没有样式CSS情况下也以一种文档格式显示，并且是容易阅读的;

搜索引擎的爬虫也依赖于HTML标记来确定上下文和各个关键字的权重，利于SEO;

使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。

9，HTML5的离线储存怎么使用，工作原理能不能解释一下？

在用户没有连网时，可以正常访问站点或应用，在用户与网络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。

10，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。

11，请描述一下cookies，sessionStorage 和localStorage 的区别？

cookie是网站为了标示用户身份而储存在用户本地终端（Client Side）上的数据（通常经过加密），cookie还可以设置有效时间

cookie数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回传递，

没次ajax请求都会吧cookie传送到后台，cookie一半用做用户登陆，后台可以根据cookie信息判断用户是否登陆状态

sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。

区别在于

存储大小：

        cookie数据大小不能超过4k。

        sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。

有期时间：

        localStorage    存储持久数据，浏览器关闭后数据不丢失除非主动删除数据；

        sessionStorage  数据在当前浏览器窗口关闭后自动删除。

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

12，iframe有那些缺点？

iframe会阻塞主页面的Onload事件；

搜索引擎的检索程序无法解读这种页面，不利于SEO;

iframe和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。

使用iframe之前需要考虑这两个缺点。如果需要使用iframe，最好是通过javascript

动态给iframe添加src属性值，这样可以绕开以上两个问题。

13，Label的作用是什么？是怎么用的？

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>

14，如何实现浏览器内多个标签页之间的通信?

WebSocket、也可以调用localstorge、cookies等本地存储方式，还可以使用页面的路有参数传递

localstorge另一个浏览上下文里被添加、修改或删除时，它都会触发一个事件，

我们通过监听事件，控制它的值来进行页面信息通信；

15，如何在页面上实现一个圆形的可点击区域？

map+area或者svg

border-radius

纯js实现 需要求一个点在不在圆上简单算法、获取鼠标坐标等等

16，title与h1的区别、b与strong的区别、i与em的区别？

title属性没有明确意义只表示是个标题，H1则表示层次明确的标题，对页面信息的抓取也有很大的影响；

strong是标明重点内容，有语气加强的含义，使用阅读设备阅读网络时：<strong>会重读，而<B>是展示强调内容。

 i内容展示为斜体，em表示强调的文本；

17，data-属性的作用是什么？

h5新增的属性        可以通过ele.dataset获取到标签上的data-\*的属性            返回一个对象

18，浏览器标准模式和怪异模式之间的区别是什么？

19，常见兼容性问题？

问题：png24位的图片在iE6浏览器上出现背景，

解决：解决方案是做成PNG8.

问题：浏览器默认的margin和padding不同。

解决：方案是加一个全局的\*{margin:0;padding:0;}来统一。

问题：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。

浮动ie产生的双倍距离 #box{ float:left; width:10px; margin:0 0 0 100px;}这种情况之下IE会产生20px的距离，

解决：解决方案是在float的标签样式控制中加入 ——\_display:inline;将其转化为行内属性。(\_这个符号只有ie6会识别)

                 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游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。

·     css

  .bb{

   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\*/

  }

问题：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：解决方法:统一通过getAttribute()获取自定义属性.

问题：IE下,even对象有x,y属性,但是没有pageX,pageY属性，Firefox下,event对象有pageX,pageY属性,但是没有x,y属性.

解决方法：（条件注释）缺点是在IE浏览器下可能会增加额外的HTTP请求数。

问题：Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示,

解决：可通过加入 CSS 属性 -webkit-text-size-adjust: none; 解决.

问题：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在具有hover和active了

解决：方法是改变CSS属性的排列顺序:L-V-H-A : a:link {} a:visited {} a:hover {} a:active {}

20，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。

21， HTML与XHTML——二者有什么区别？

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

22，<img>的title和alt有什么区别？

title是global attributes之一，用于为元素提供附加的advisory information。通常当鼠标滑动到元素上的时候显示。

alt是<img>的特有属性，是图片内容的等价描述，用于图片无法加载时显示、读屏器阅读图片。可提图片高可访问性，除了纯装饰图片外都必须设置有意义的值，搜索引擎会重点分析。

CSS篇

1，介绍一下标准的CSS的盒子模型？低版本IE的盒子模型有什么不同的？

有两种， IE 盒子模型、W3C 盒子模型；

盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border)；

区  别： IE的content部分把 border 和 padding计算了进去；

在CSS3中引入了box-sizing属性, 它可以允许改变默认的CSS盒模型对元素宽高的计算方式.

共包括两个选项：

content-box：标准盒模型，CSS定义的宽高只包含content的宽高

border-box：IE盒模型，CSS定义的宽高包括了content，padding和border

2，CSS选择符有哪些？哪些属性可以继承？

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）

可继承的样式： font-size font-family color, UL LI DL DD DT;

不可继承的样式：border padding margin width height ;

3，CSS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

4，CSS3新增伪类有那些？

举例：

p:first-of-type 选择属于其父元素的首个 <p>元素的每个 <p>元素。

p:last-of-type  选择属于其父元素的最后 <p>元素的每个 <p>元素。

p:only-of-type  选择属于其父元素唯一的 <p>元素的每个 <p>元素。

p:only-child        选择属于其父元素的唯一子元素的每个 <p>元素。

p:nth-child(2)  选择属于其父元素的第二个子元素的每个 <p>元素。

:after          在元素之前添加内容,也可以用来做清除浮动。

:before         在元素之后添加内容

:enabled

:disabled       控制表单控件的禁用状态。

:checked        单选框或复选框被选中。

5，如何居中div？

水平垂直居中问题，在css中margin: 0 auto;可以实现水平居中，但是在垂直居中方面一直没有特定的属性，直到css3的出现，有了弹性盒，可以通过设置弹性盒直接设置居中位置，做浏览器兼容的话可以通过使用一些hack处理 负margin方法，table-cell方法，位移方法

1，负margin方法：

CSS代码：

.container{

        width: 500px;

        height: 400px;

        border: 2px solid #379;

        position: relative;

}

.inner{

        width: 480px;

        height: 380px;

        background-color: #746;

        position: absolute;

        top: 50%;

        left: 50%;

        margin-top: -190px; /\*height的一半\*/

        margin-left: -240px; /\*width的一半\*/

}

HTML代码：

<div class="container">     <div class="inner"></div> </div>

这里，我们首先用top:50%和left:50%让inner的坐标原点（左上角）移动到container的中心，然后再利用负margin让它往左偏移自身宽的一半，再往上偏移自身高的一半，这样inner的中心点就跟container的中心点对齐了。

2，table－cell方法

CSS代码：

div{

     width: 300px;

     height: 300px;

     border: 3px solid #555;

     display: table-cell;

     vertical-align: middle;

     text-align: center;

}

img{

     vertical-align: middle;

}

HTML代码：

<div>     <img src="mm.jpg"> </div>

通过将盒子转换为table元素，table元素本身是可以通过属性来控制位置，div上面的vertical-align: middle是控制垂直方向上的居中的，而text-align: center是控制水平方向的

3，弹性盒子法

CSS代码：

.container{

       width: 300px;

       height: 200px;

       border: 3px solid #546461;

       display: -webkit-flex;

       display: flex;

       -webkit-align-items: center;

       align-items: center;

       -webkit-justify-content: center;

       justify-content: center;

}

.inner{

       border: 3px solid #458761;

       padding: 20px;

}

HTML代码：

<div class="container">

     <div class="inner">

         我在容器中水平垂直居中

     </div>

</div>

align-items控制垂直方向的居中，justify-content控制水平方向的居中。这是CSS3的新方法

4，位移方法

位移方法和margin定位方法一样，只不过吧margin改成了位移不需要计算一半是多少，直接 transform:translate(-50%,--50%)

6，display有哪些值？说明他们的作用。

block              象块类型元素一样显示。

none               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。

table               此元素会作为块级表格来显示

inherit            规定应该从父元素继承 display 属性的值

7，position的值relative和absolute定位原点是？

absolute  生成绝对定位的元素，相对于值不为 static的第一个父元素进行定位。

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，相对于其正常位置进行定位。

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。

8，CSS3有哪些新特性？

新增各种CSS选择器  （: not(.input)：所有 class 不是“input”的节点）

圆角           （border-radius:8px）

多列布局        （multi-column layout）

阴影和反射        （Shadow\Reflect）

文字特效      （text-shadow、）

文字渲染      （Text-decoration）

线性渐变      （gradient）

旋转          （transform）

增加了旋转,缩放,定位,倾斜,动画，多背景

transform:\scale(0.85,0.90)\ translate(0px,-30px)\ skew(-9deg,0deg)\Animation:

9，用纯CSS创建一个三角形的原理是什么？

把上、左、右三条边隐藏掉（颜色设为 transparent）

#demo {

        width: 0;

        height: 0;

        border-width: 20px;

        border-style: solid;

        border-color: transparent transparent red transparent;

}

10，一个满屏品字布局如何设计?

简单的方式：

上面的div宽100%，

下面的两个div分别宽50%，

然后用float或者inline使其不换行即可

11，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？

行框的排列会受到中间空白（回车\空格）等的影响，因为空格也属于字符,这些空白也会被应用样式，占据空间，所以会有间隔，把字符大小设为0，就没有空格了。

12，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： \* {padding: 0; margin: 0;} （强烈不建议）

淘宝的样式初始化代码：

                  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; }

                  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; }

13，absolute的containing block(容器块)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)，根元素也是initial containing block

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

(W3C CSS 2.1 规范中的一个概念,它是一个独立容器，决定了元素如何对其内容进行定位,以及与其他元素的关系和相互作用。）

 一个页面是由很多个 Box 组成的,元素的类型和 display 属性,决定了这个 Box 的类型。

不同类型的 Box,会参与不同的 Formatting Context（决定如何渲染文档的容器）,因此Box内的元素会以不同的方式渲染,也就是说BFC内部的元素和外部的元素不会互相影响。

                  创建规则：

                  根元素

                  浮动元素（float不是none）

                  绝对定位元素（position取值为absolute或fixed）

                  display取值为inline-block,table-cell, table-caption,flex, inline-flex之一的元素

                  overflow不是visible的元素

                  作用：

                  可以包含浮动元素

                  不被浮动元素覆盖

                  阻止父子元素的margin折叠

15，css定义的权重

以下是权重的规则：标签的权重为1，class的权重为10，id的权重为100，以下例子是演示各种定义的权重值：

                  /\*权重为1\*/            div{}

                  /\*权重为10\*/          .class1{}

                  /\*权重为100\*/       #id1{}

                  /\*权重为100+1=101\*/         #id1 div{}

                  /\*权重为10+1=11\*/              .class1 div{}

                  /\*权重为10+10+1=21\*/      .class1 .class2 div{}

如果权重相同，则最后定义的样式会起作用，但是应该避免这种情况出现

16，请解释一下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

浮动元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留。

1.使用空标签清除浮动。

这种方法是在所有浮动标签后面添加一个空标签 定义css clear:both. 弊端就是增加了无意义标签。

2.使用overflow。

给包含浮动元素的父标签添加css属性 overflow:auto; zoom:1; zoom:1用于兼容IE6。

3.使用after伪对象清除浮动。

该方法只适用于非IE浏览器。具体写法可参照以下示例。使用中需注意以下几点。一、该方法中必须为需要清除浮动元素的伪对象中设置 height:0，否则该元素会比实际高出若干像素；

16，浮动元素引起的问题和解决办法？

浮动元素引起的问题：

（1）父元素的高度无法被撑开，影响与父元素同级的元素

（2）与浮动元素同级的非浮动元素（内联元素）会跟随其后

（3）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

解决方法：

使用CSS中的clear:both;属性来清除元素的浮动可解决2、3问题，对于问题1，添加如下样式，给父元素添加clearfix样式：

.clearfix:after{content: ".";display: block;height: 0;clear: both;visibility: hidden;}

.clearfix{display: inline-block;} /\* for IE/Mac \*/

清除浮动的几种方法：

1，额外标签法，<div style="clear:both;"></div>（缺点：不过这个办法会增加额外的标签使HTML结构看起来不够简洁。）

2，使用after伪类

#parent:after{

        content:".";             height:0; visibility:hidden;                      display:block;         clear:both;

 }

3,浮动外部元素

4,设置overflow为hidden或者auto

17，移动端的布局用过媒体查询吗？

css的媒体查询允许通过@media标签为特定媒体的浏览器设定样式，其中包含众多筛选，功能强大。

18，使用CSS 预处理器吗？喜欢那个？

SASS (SASS、LESS没有本质区别，只因为团队前端都是用的SASS)，可以使用sass和less对css做模块化开发，定制样式的组件

19，CSS优化、提高性能的方法有哪些？

将样式表放到页面顶部

不使用CSS表达式

不使用@import

不使用IE的Filter

20，元素竖向的百分比设定是相对于容器的高度吗？

是的，元素的百分比设置一般是根据父级元素的宽高决定的，如果父级元素没有宽高百分比是不起作用的

21，什么是响应式设计？响应式设计的基本原理是什么？

响应式设计就是为了实现根据不同设备环境自动响应及调整网页布局的一种设计方案

基本原理就是利用css的媒体查询功能更具不同屏幕的大小，向下兼容设备、移动优先，达到响应的效果

22，::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。

双冒号是在当前规范中引入的，用于区分伪类和伪元素。但是伪类兼容现存样式，浏览器需要同时支持旧的伪类，比如:first-line、:first-letter、:before、:after等。

对于CSS2之前已有的伪元素，比如:before和:after，单冒号和双冒号的写法::before和::after作用是一样的。

如果只需要兼容webkit、firefox、opera等浏览器，建议对于伪元素采用双冒号的写法，如果不得不兼容IE浏览器，还是用CSS2的单冒号写法比较安全。

23，display:inline-block 什么时候会显示间隙？

移除空格、使用margin负值、使用font-size:0、letter-spacing、word-spacing

24，png、jpg、gif 这些图片格式解释一下，分别什么时候用？

 GIF： 8位像素，256色                    无损压缩                  支持简单动画         支持boolean透明     适合简单动画

JPEG：颜色限于256          有损压缩                  可控制压缩质量                       不支持透明              适合照片

PNG：有PNG8和truecolor PNG    PNG8类似GIF颜色上限为256，文件小，支持alpha透明度，无动画、适合图标、背景、按钮

25，是CSS 预处理器/ 后处理器？

- 预处理器例如：LESS、Sass、Stylus，用来预编译Sass或less，增强了css代码的复用性，还有层级、mixin、变量、循环、函数等，具有很方便的UI组件模块化开发能力，极大的提高工作效率。

- 后处理器例如：PostCSS，通常被视为在完成的样式表中根据CSS规范处理CSS，让其更有效；目前最常做的是给CSS属性添加浏览器私有前缀，实现跨浏览器兼容性的问题。

26，css sprite是什么,有什么优缺点。

概念：将多个小图片拼接到一个图片中。通过background-position和元素尺寸调节需要显示的背景图案

优点：

减少HTTP请求数，极大地提高页面加载速度

增加图片信息重复度，提高压缩比，减少图片大小

更换风格方便，只需在一张或几张图片上修改颜色或样式即可实现

缺点：

图片合并麻烦

维护麻烦，修改一个图片可能需要从新布局整个图片，样式

27，你用过栅格系统吗？

现在大部分UI框架都内置有删格化系统，常用bootstrap中的，bootstap它提供了一套响应式，移动优先的流式栅格系统(grid system)，将屏幕分成12列来实现响应式的。它的实现原理非常简单，Media Query加上float布局，也研究过bootstrap的源码，在这个过程中也对预处理器有了更深的体会，删格系统这块在bootstrap中sass源码使用的循环生成，less使用的递归生成，我自己也单独封装过一个删格系统

28，渐进增强和优雅降级

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、交互等改进和追加功能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。

29，什么是FOUC（无样式内容闪烁）？你如何来避免FOUC？

FOUC - Flash Of Unstyled Content 文档样式闪烁

<style type="text/css" media="all">@import "../fouc.css";</style>

而引用CSS文件的@import就是造成这个问题的罪魁祸首。IE会先加载整个HTML文档的DOM，然后再去导入外部的CSS文件，因此，在页面DOM加载完成到CSS导入完成中间会有一段时间页面上的内容是没有样式的，这段时间的长短跟网速，电脑速度都有关系。

解决方法简单的出奇，只要在<head>之间加入一个<link>或者<script>元素就可以了。

30，display: none;与visibility: hidden;的区别

联系：它们都能让元素不可见

区别：

display:none;会让元素完全从渲染树中消失，渲染的时候不占据任何空间；visibility: hidden;不会让元素从渲染树消失，渲染师元素继续占据空间，只是内容不可见

display: none;是非继承属性，子孙节点消失由于元素从渲染树消失造成，通过修改子孙节点属性无法显示；visibility: hidden;是继承属性，子孙节点消失由于继承了hidden，通过设置visibility: visible;可以让子孙节点显式

修改常规流中元素的display通常会造成文档重排。修改visibility属性只会造成本元素的重绘。

读屏器不会读取display: none;元素内容；会读取visibility: hidden;元素内容

31，常见的布局方式

常见的布局这么几种单列水平居中布局, 一列定宽一列自适应布局, 两列定宽一列自适应布局, 两侧定宽中间自适应三列布局。

一列定宽一列自适应

定位布局

左边的宽度写死，右边盒子使用定位拉伸法实现，left等于左边盒子的宽度，right等于0

.left-box{ width:200px;   }

.right-box{  position:absolute; left:200px; right:0; }

或者左边的定位写死宽度，右边的写padding-left等于左边的宽度（常用一点）

.left-box{ width:200px;  position: fixed; height:100%; }

.right-box{  padding-left:200px;}

浮动布局

左边的宽度写死并且浮动，右边盒子写overflow:hidden;利用的是创建一个新的BFC（块级格式化上下文）来防止文字环绕的原理来实现的。BFC就是一个相对独立的布局环境，它内部元素的布局不受外面布局的影响

.left-box {

        width: 200px;

        height: 150px;

        float: left;

        background: #f22;

}

.right-box {

        height: 200px;

        overflow: hidden;

        background: #cff;

 }

两列定宽一列自适应上面的布局方式同样适用

常见的三列布局一般使用圣杯布局和双飞翼布局。

圣杯布局和双飞翼布局

两者都属于三列布局，是一种很常见的页面布局方式，

三列一般分别是子列、主列和附加列，其中子列一般是居左的导航，且宽度固定；主列是居中的主要内容，宽度自适应；附加列一般是广告等额外信息，居右且宽度固定。

圣杯布局

<div class="container">

    <div class="main"></div>

    <div class="sub"></div>

    <div class="extra"></div>

</div>

.container {

    padding-left: 210px;

    padding-right: 190px;

}

.main {

    float: left;

    width: 100%;

    height: 300px;

}

.sub {

    position: relative;

    left: -210px;

    float: left;

    width: 200px;

    height: 300px;

    margin-left: -100%;

}

.extra {

    position: relative;

    right: -190px;

    float: left;

    width: 180px;

    height: 300px;

    margin-left: -180px;

}

双飞翼布局

<div class="main-wrapper">

    <div class="main"></div>

</div>

<div class="sub"></div>

<div class="extra"></div>

.main-wrapper {

    float: left;

    width: 100%;

}

.main {

    height: 300px;

    margin-left: 210px;

    margin-right: 190px;

    background-color: rgba(255, 0, 0, .5);

}

.sub {

    float: left;

    width: 200px;

    height: 300px;

    margin-left: -100%;

    background-color: rgba(0, 255, 0, .5);

}

.extra {

    float: left;

    width: 180px;

    height: 300px;

    margin-left: -180px;

    background-color: rgba(0, 0, 255, .5);

}

俩种布局方式都是把主列放在文档流最前面，使主列优先加载。

两种布局方式在实现上也有相同之处，都是让三列浮动，然后通过负外边距形成三列布局。

两种布局方式的不同之处在于如何处理中间主列的位置：圣杯布局是利用父容器的左、右内边距定位；双飞翼布局是把主列嵌套在div后利用主列的左、右外边距定位。

其他的话还有

flex布局

Flexbox又叫弹性盒模型。它可以简单使用一个元素居中（包括水平垂直居中），可以让扩大和收缩元素来填充容器的可利用空间，可以改变源码顺序独立布局，以及还有其他的一些功能。

Flex布局是我最喜欢的布局，合理使用它能够大大减少布局方面的工作, 例如以上列举的三列布局也可以使用flex轻松实现。此外在移动端使用flex也比较常见。

响应式布局

网页可以自动识别设备屏幕宽度，根据不同的宽度采用不同的CSS的样式，从而达到兼容各种设备的效果。

响应式布局使用媒体查询(CSS3 Media Queries), 根据不同屏幕分辨率采用不同CSS规则

流式布局

流式布常见的就是bootstrap了它提供了一套响应式，移动优先的流式栅格系统(grid system)，将屏幕分成12列来实现响应式的。它的实现原理非常简单，Media Query加上float布局

32，说一下CSS的hack

由于不同厂商的流览器或某浏览器的不同版本（如IE6-IE11,Firefox/Safari/Opera/Chrome等），对CSS的支持、解析不一样，导致在不同浏览器的环境中呈现出不一致的页面展现效果。这时，我们为了获得统一的页面效果，就需要针对不同的浏览器或不同版本写特定的CSS样式，我们把这个针对不同的浏览器/不同版本写相应的CSS code的过程，叫做CSS hack

**CSS Hack**的方法

条件注释法

<!--[if IE]>

这段文字只在IE浏览器显示

<![endif]—>

类内属性前缀法

.hack{

    background-color:red; /\* All browsers \*/

    background-color:blue !important;/\* All browsers but IE6 \*/

    \*background-color:black; /\* IE6, IE7 \*/

    +background-color:yellow;/\* IE6, IE7\*/

    background-color:gray\9; /\* IE6, IE7, IE8, IE9, IE10 \*/

    background-color:purple\0; /\* IE8, IE9, IE10 \*/

    background-color:orange\9\0;/\*IE9, IE10\*/

    \_background-color:green; /\* Only works in IE6 \*/

}

媒体查询

@**media** \0screen\,screen\9 {

  .hack{

    /\* IE 6 7 8 \*/

  }

}

@**media** screen and (-webkit-min-device-pixel-ratio:0) {

   .demo {

      /\* Webkit内核 \*/

    }

}

**33**，说一下居中

水平垂直居中问题，在css中margin: 0 auto;可以实现水平居中，但是在垂直居中方面一直没有特定的属性，直到css3的出现，有了弹性盒，可以通过设置弹性盒直接设置居中位置，做浏览器兼容的话可以通过使用一些hack处理 负margin方法， table-cell方法，位移方法

负**margin**方法

我们首先用top:50%和left:50%让inner的坐标原点（左上角）移动到container的中心，然后再利用负margin让它往左偏移自身宽的一半，再往上偏移自身高的一半，这样inner的中心点就跟container的中心点对齐了。

**table**－**cell**方法

通过将盒子转换为table元素，table元素本身是可以通过属性来控制位置，div上面的vertical-align: middle是控制垂直方向上的居中的，而text-align: center是控制水平方向的

弹性盒子法

align-items控制垂直方向的居中，justify-content控制水平方向的居中。这是CSS3的新方法

位移方法

位移方法和margin定位方法一样，只不过吧margin改成了位移不需要计算一半是多少，直接 transform:translate(-50%,--50%)

**34**，样式继承

可继承的样式font-size font-family color text-indent

**35**，**link**和**@import**的区别

link属于XHTML标签，而@import是CSS提供的

页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载

@import只在IE5以上才能识别，而link是XHTML标签，无兼容问题

link方式的样式的权重 高于@import的权重

**36**，**position**属性

static：默认的属性值，按照标准流（包括浮动方式）进行布局。

relative：称为相对定位，使用相对的盒子的位置常以标准量的排版方式为基础，然后使盒子相对于它在原本的标准位置偏移指定的距离.相对定位仍在标准流中，它对父块和兄弟块盒子没有任何影响。

absolute：绝对定位，盒子的位置以它的包含框为基准进行偏移。绝对定位从标准流中脱离，并且以它最近的一个已经定位的祖先元素为基准进行定位。没有已经定位的祖先元素，则以浏览器窗口为基准进行定位。

inherit：规定从父元素继承 position 属性的值。

fixed：固定定位，与绝对定位类似，以浏览器窗口为基准进行定位，拖动浏览器窗口的滚动条时，位置保持不变。

sticky：粘性定位，粘性定位是相对定位和固定定位的混合。元素在跨越特定阈值前为相对定位，之后为固定定位。但是这个属性兼容性不太好，ios上支持度相对好一些

JavaScript篇

**1**，介绍**js**的基本数据类型。

Undefined、Null、Boolean、Number、String

**2**，介绍**js**有哪些内置对象？

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

3，JavaScript原型，原型链 ? 有什么特点？

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

关系：instance.constructor.prototype = instance.\_\_proto\_\_

特点：

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

当我们需要一个属性的时，Javascript引擎会先看当前对象中是否有这个属性， 如果没有的话，就会查找他的Prototype对象是否有这个属性，如此递推下去，一直检索到 Object 内建对象。

4，JavaScript有几种类型的值？，你能画一下他们的内存图吗？

栈：原始数据类型（Undefined，Null，Boolean，Number、String）

堆：引用数据类型（对象、数组和函数）

两种类型的区别是：存储位置不同；

1，原始数据类型直接存储在栈(stack)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；

2，引用数据类型存储在堆(heap)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

**5**，JavaScript继承的几种实现方式？

1，构造函数继承，使用call和apply两个方法的特性可以实现，改变方法中的this

2，原型链继承

3，组合式继承

6，javascript创建对象的几种方式？

javascript创建对象简单的说,无非就是使用内置对象或各种自定义对象，当然还可以用JSON；但写法有很多种，也能混合使用。

1、对象字面量的方式       person={firstname:"Mark",lastname:"Yun",age:25,eyecolor:"black"};

2、用function来模拟无参的构造函数

            function Person(){}

                        var person=new Person();//定义一个function，如果使用new"实例化",该function可以看作是一个Class

                        person.name=“Mark";

                        person.age="25";

                        person.work=function(){

                        alert(person.name+" hello...");

            }

            person.work();

3、用function来模拟参构造函数来实现（用this关键字定义构造的上下文属性）

            function Pet(name,age,hobby){

                        this.name=name;//this作用域：当前对象

                        this.age=age;

                        this.hobby=hobby;

                        this.eat=function(){

                                    alert("我叫"+this.name+",我喜欢"+this.hobby+",是个程序员");

                         }

             }

            var maidou =new Pet("麦兜",25,"coding");//实例化、创建对象

            maidou.eat();//调用eat方法

4、用工厂方式来创建（内置对象）

                       var wcDog =new Object();

                       wcDog.name="旺财";

                       wcDog.age=3;

                       wcDog.work=function(){

                                    alert("我是"+wcDog.name+",汪汪汪......");

                         }

            wcDog.work();

5、用原型方式来创建

            function Dog(){ }

            Dog.prototype.name="旺财";

            Dog.prototype.eat=function(){alert(this.name+"是个吃货");}

            var wangcai =new Dog();

            wangcai.eat();

5、用混合方式来创建

            function Car(name,price){

                        this.name=name;

                        this.price=price;

            }

            Car.prototype.sell=function(){

                        alert("我是"+this.name+"，我现在卖"+this.price+"万元");

             }

             var camry =new Car("凯美瑞",27);

            camry.sell();

7，Javascript作用链域?

首先在js中有作用域的概念，值得就是一个变量的活动范围，分为全局作用域和局部作用域，全局作用域指的是window，局部作用域指的是每一个函数内部，在作用域中查找一个变量首先在自己当前作用域查找找不到向上级查找，逐层向上找到window为止，找不到就会抛出一个错误，这个查找的过程就叫作用域链，作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。作用域链有一个需要要注意的问题就是变量提升，当一个变量的使用在定义之前的时候就会得到一个undefined值，在es6中则不会出现这个问题，es6不允许在定义之前使用

8，谈谈This对象的理解。

this分为几个不同的使用场景，在function中this指的的是window，如果是实用new 调用的话this指的是当前的实例化对象，在事件调用函数中this指的调用事件的window特殊的是在IE中的attachEvent中的this总是指向全局对象Window；，在定时器中this指的是window，在es6中有一个箭头函数，在箭头函数中this永远指向的是父级对象

9，什么是window对象? 什么是document对象?

window:它是一个顶层对象,而不是另一个对象的属性，即浏览器的窗口。

document:代表整个HTML 文档,可用来访问页面中的所有元素

Window 对象表示当前浏览器的窗口，是JavaScript的顶级对象。我们创建的所有对象、函数、变量都是 Window 对象的成员。

Window 对象的方法和属性是在全局范围内有效的。

Document 对象是 HTML 文档的根节点与所有其他节点（元素节点，文本节点，属性节点, 注释节点）

Document 对象使我们可以通过脚本对 HTML 页面中的所有元素进行访问

Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问

10，null，undefined 的区别？

null        表示一个对象被定义了，值为“空值”；

undefined   表示不存在这个值。

typeof undefined                      //"undefined"

undefined :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义。当尝试读取时会返回 undefined；

例如变量被声明了，但没有赋值时，就等于undefined

typeof null        //"object"

null : 是一个对象(空对象, 没有任何属性和方法)；

例如作为函数的参数，表示该函数的参数不是对象；

注意：

在验证null时，一定要使用　=== ，因为 == 无法分别 null 和　undefined

undefined表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法是：

1）变量被声明了，但没有赋值时，就等于undefined。

2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。

3）对象没有赋值的属性，该属性的值为undefined。

4）函数没有返回值时，默认返回undefined。

null表示"没有对象"，即该处不应该有值。典型用法是：

1） 作为函数的参数，表示该函数的参数不是对象。

2） 作为对象原型链的终点。

11，事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡？

1. 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。

2. 事件处理机制：IE是事件冒泡、Firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件；

3. ev.stopPropagation();（旧ie的方法 ev.cancelBubble = true;）

12，什么是闭包（closure），为什么要用它？

闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量,利用闭包可以突破作用链域，将函数内部的变量和方法传递到外部。

闭包的特性：

1，函数内再嵌套函数

2，内部函数可以引用外层的参数和变量

3，参数和变量不会被垃圾回收机制回收

闭包的使用场景

常见的闭包的使用场景就是模块化，用来做模块内部的实现通过接口的扩展贡其他模块使用

在就是闭包可以用来缓存值，减少不必要的技术，例如vue里面的计算属性

13，javascript 代码中的"use strict";是什么意思 ? 使用它区别是什么？

use strict是一种ECMAscript 5 添加的（严格）运行模式,这种模式使得 Javascript 在更严格的条件下运行,

使JS编码更加规范化的模式,消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为。

默认支持的糟糕特性都会被禁用，比如不能用with，也不能在意外的情况下给全局变量赋值;

全局变量的显示声明,函数必须声明在顶层，不允许在非函数代码块内声明函数,arguments.callee也不允许使用；

消除代码运行的一些不安全之处，保证代码运行的安全,限制函数中的arguments修改，严格模式下的eval函数的行为和非严格模式的也不相同;

提高编译器效率，增加运行速度；

为未来新版本的Javascript标准化做铺垫。

14，如何判断一个对象是否属于某个类？

1，使用instanceof

if(a instanceof Person){ alert('yes’); }

15，Javascript中，有一个函数，执行时对象查找时，永远不会去查找原型，这个函数是？

hasOwnProperty

 javaScript中hasOwnProperty函数方法是返回一个布尔值，指出一个对象是否具有指定名称的属性。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。

使用方法：

object.hasOwnProperty(proName)

其中参数object是必选项。一个对象的实例。

proName是必选项。一个属性名称的字符串值。

如果 object 具有指定名称的属性，那么JavaScript中hasOwnProperty函数方法返回 true，反之则返回 false。

16，JSON 的了解？

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。

它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小

如：{"age":"12", "name":"back"}

JSON字符串转换为JSON对象:

var obj =eval('('+ str +')');

var obj = str.parseJSON();

var obj = JSON.parse(str);

JSON对象转换为JSON字符串：

var last=obj.toJSONString();

var last=JSON.stringify(obj);

17，documen.write和 innerHTML的区别

document.write只能重绘整个页面

innerHTML可以重绘页面的一部分

18，DOM操作——怎样添加、移除、移动、复制、创建和查找节点?

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

19，.call() 和 .apply() 的区别？

20，数组和对象有哪些原生方法，列举一下？

21，字符串有哪些原生方法，列举一下？

22，那些操作会造成内存泄漏？

1，使用闭包的时候如果在闭包变量中保存了大量的dom结构而且不去使用长期存在的时候

2，内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

3，垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

4，setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

5，闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

23，页面重构怎么操作？

网站重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。

也就是说是在不改变UI的情况下，对网站进行优化，在扩展的同时保持一致的UI。

对于传统的网站来说重构通常是：

表格(table)布局改为DIV+CSS

使网站前端兼容于现代浏览器(针对于不合规范的CSS、如对IE6有效的)

对于移动平台的优化

针对于SEO进行优化

深层次的网站重构应该考虑的方面

减少代码间的耦合

让代码保持弹性

严格按规范编写代码

设计可扩展的API

代替旧有的框架、语言(如VB)

增强用户体验

通常来说对于速度的优化也包含在重构中

压缩JS、CSS、image等前端资源(通常是由服务器来解决)

程序的性能优化(如数据读写)

采用CDN来加速资源加载

对于JS DOM的优化

HTTP服务器的文件缓存

24，列举IE与其他浏览器不一样的特性？

1、事件不同之处：

    1-1，触发事件的元素被认为是目标（target）。而在 IE 中，目标包含在 event 对象的 srcElement 属性；

    1-2，获取字符代码、如果按键代表一个字符（shift、ctrl、alt除外），IE 的 keyCode 会返回字符代码（Unicode），DOM 中按键的代码和字符是分离的，要获取字符代码，需要使用 charCode 属性；

    1-3，阻止某个事件的默认行为，IE 中阻止某个事件的默认行为，必须将 returnValue 属性设置为 false，Mozilla 中，需要调用 preventDefault() 方法；

    1-4，停止事件冒泡，IE 中阻止事件进一步冒泡，需要设置 cancelBubble 为 true，Mozzilla 中，需要调用 stopPropagation()；

25，什么叫优雅降级和渐进增强？

优雅降级：Web站点在所有新式浏览器中都能正常工作，如果用户使用的是老式浏览器，则代码会针对旧版本的IE进行降级处理了,使之在旧式浏览器上以某种形式降级体验却不至于完全不能用。

如：border-shadow

渐进增强：从被所有浏览器支持的基本功能开始，逐步地添加那些只有新版本浏览器才支持的功能,向页面增加不影响基础浏览器的额外样式和功能的。当浏览器支持时，它们会自动地呈现出来并发挥作用。

如：默认使用flash上传，但如果浏览器支持 HTML5 的文件上传功能，则使用HTML5实现更好的体验；

26，是否了解公钥加密和私钥加密。

一般情况下是指私钥用于对数据进行签名，公钥用于对签名进行验证;

 HTTP网站在浏览器端用公钥加密敏感数据，然后在服务器端再用私钥解密。

27，对Node的优点和缺点提出了自己的看法？

\*（优点）因为Node是基于事件驱动和无阻塞的，所以非常适合处理并发请求，因此构建在Node上的代理服务器相比其他技术实现（如Ruby）的服务器表现要好得多。此外，与Node代理服务器交互的客户端代码是由javascript语言编写的，因此客户端和服务器端都用同一种语言编写，这是非常美妙的事情。

\*（缺点）Node是一个相对新的开源项目，所以不太稳定，它总是一直在变，而且缺少足够多的第三方库支持。看起来，就像是Ruby/Rails当年的样子。

28，解释一下事件代理

事件代理的原理其实就和作用域链的原理差不多，但是事件代理是利用事件的冒泡原理来实现的，事件代理就是通过给祖先元素添加事件，通过事件目标对象开始向上查找找到匹配的子节点为止，如果找不到则到绑定事件的那个祖先元素为止，找到了就触发事件，并且可以通过js中call和apply来改变触发事件函数中的this为当前绑定节点，也是通过一层一层逐层向上的方式进行匹配查找而触发对应事件，好处就是可以使后添加的dom元素也同样有之前存在元素的事件，jquery中可以使用on，delegate，live实现的，不过在jquery1.7版本以后吧live给废除了，原因就是live绑定事件的祖先元素是整个html页面的根节点，所以性能消耗比较大，在后边的版本中给删除了，使用on，delegate代替

优点：

可以减少事件注册，节省大量内存占用

可以将事件应用于动态添加的子元素上

缺点： 使用不当会造成事件在不应该触发时触发

29，Javascript垃圾回收方法

标记清除（mark and sweep）

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

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

引用计数(reference counting)

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

            在IE中虽然JavaScript对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的， 也就是说只要涉及BOM及DOM就会出现循环引用问题。

30，说说严格模式的限制

严格模式主要有以下限制：

变量必须声明后再使用

函数的参数不能有同名属性，否则报错

不能使用with语句

不能对只读属性赋值，否则报错

不能使用前缀0表示八进制数，否则报错

不能删除不可删除的属性，否则报错

不能删除变量delete prop，会报错，只能删除属性delete global[prop]

eval不会在它的外层作用域引入变量

eval和arguments不能被重新赋值

arguments不会自动反映函数参数的变化

不能使用arguments.callee

不能使用arguments.caller

禁止this指向全局对象

不能使用fn.caller和fn.arguments获取函数调用的堆栈

增加了保留字（比如protected、static和interface）

设立"严格模式"的目的，主要有以下几个：

消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

消除代码运行的一些不安全之处，保证代码运行的安全；

提高编译器效率，增加运行速度；

为未来新版本的Javascript做好铺垫。

注：经过测试IE6,7,8,9均不支持严格模式。

31，attribute和property的区别是什么？

attribute是dom元素在文档中作为html标签拥有的属性；

property就是dom元素在js中作为对象拥有的属性。

所以：

对于html的标准属性来说，attribute和property是同步的，是会自动更新的，

但是对于自定义的属性来说，他们是不同步的，

32，offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别，

offsetWidth/offsetHeight返回值包含content + padding + border，效果与e.getBoundingClientRect()相同

clientWidth/clientHeight返回值只包含content + padding，如果有滚动条，也不包含滚动条

scrollWidth/scrollHeight返回值包含content + padding + 溢出内容的尺寸

33，focus/blur与focusin/focusout的区别与联系

focus/blur不冒泡，focusin/focusout冒泡

focus/blur兼容性好，focusin/focusout在除FireFox外的浏览器下都保持良好兼容性，如需使用事件托管，可考虑在FireFox下使用事件捕获elem.addEventListener('focus', handler, true)

可获得焦点的元素：

window

链接被点击或键盘操作

表单空间被点击或键盘操作

设置tabindex属性的元素被点击或键盘操作

34，mouseover/mouseout与mouseenter/mouseleave的区别与联系

mouseover/mouseout是标准事件，所有浏览器都支持；mouseenter/mouseleave是IE5.5引入的特有事件后来被DOM3标准采纳，现代标准浏览器也支持

mouseover/mouseout是冒泡事件；mouseenter/mouseleave不冒泡。需要为多个元素监听鼠标移入/出事件时，推荐mouseover/mouseout托管，提高性能

标准事件模型中event.target表示发生移入/出的元素,vent.relatedTarget对应移出/如元素；在老IE中event.srcElement表示发生移入/出的元素，event.toElement表示移出的目标元素，event.fromElement表示移入时的来源元素

**35**，检测浏览器版本版本有哪些方式？

根据 navigator.userAgent // UA.toLowerCase().indexOf('chrome')

根据 window 对象的成员 // 'ActiveXObject' in window

**36**，描述浏览器的渲染过程，**DOM**树和渲染树的区别？

浏览器的渲染过程：

解析HTML构建 DOM(DOM树)，并行请求 css/image/js

CSS 文件下载完成，开始构建 CSSOM(CSS树)

CSSOM 构建结束后，和 DOM 一起生成 Render Tree(渲染树)

布局(Layout)：计算出每个节点在屏幕中的位置

显示(Painting)：通过显卡把页面画到屏幕上

**DOM**树和渲染树的区别：

DOM树与HTML标签一一对应，包括head和隐藏元素

渲染树不包括head和隐藏元素，大段文本的每一个行都是独立节点，每一个节点都有对应的css属性

**37**，重绘和回流（重排）的区别和关系？

重绘：当渲染树中的元素外观（如：颜色）发生改变，不影响布局时，产生重绘

回流：当渲染树中的元素的布局（如：尺寸、位置、隐藏/状态状态）发生改变时，产生重绘回流

注意：JS获取Layout属性值（如：offsetLeft、scrollTop、getComputedStyle等）也会引起回流。因为浏览器需要通过回流计算最新值

回流必将引起重绘，而重绘不一定会引起回流

**38**，如何最小化重绘**(repaint)**和回流**(reflow)**？

需要要对元素进行复杂的操作时，可以先隐藏(display:"none")，操作完成后再显示

需要创建多个DOM节点时，使用DocumentFragment创建完后一次性的加入document

缓存Layout属性值，如：var left = elem.offsetLeft; 这样，多次使用 left 只产生一次回流

尽量避免用table布局（table元素一旦触发回流就会导致table里所有的其它元素回流）

避免使用css表达式(expression)，因为每次调用都会重新计算值（包括加载页面）

尽量使用 css 属性简写，如：用 border 代替 border-width, border-style, border-color

批量修改元素样式：elem.className 和 elem.style.cssText 代替 elem.style.xxx

**39**，**script** 的位置是否会影响首屏显示时间？

在解析 HTML 生成 DOM 过程中，js 文件的下载是并行的，不需要 DOM 处理到 script 节点。因此，script的位置不影响首屏显示的开始时间。

浏览器解析 HTML 是自上而下的线性过程，script作为 HTML 的一部分同样遵循这个原则

因此，script 会延迟 DomContentLoad，只显示其上部分首屏内容，从而影响首屏显示的完成时间

**40**，介绍**DOM0**，**DOM2**，**DOM3**事件处理方式区别

DOM0级事件处理方式：

    btn.onclick = func;

    btn.onclick = null;

DOM2级事件处理方式：

    btn.addEventListener('click', func, false);

    btn.removeEventListener('click', func, false);

    btn.attachEvent("onclick", func);

    btn.detachEvent("onclick", func);

DOM3级事件处理方式：

    eventUtil.addListener(input, "textInput", func);

    eventUtil 是自定义对象，textInput 是DOM3级事件

**41**，事件的三个阶段

捕获、目标、冒泡

**js**的冒泡**(Bubbling Event)**和捕获**(Capture Event)**的区别

冒泡型事件：事件按照从最特定的事件目标到最不特定的事件目标(document对象)的顺序触发。

捕获型事件(event capturing)：事件从最不精确的对象(document 对象)开始触发，然后到最精确(也可以在窗口级别捕获事件，不过必须由开发人员特别指定)。

DOM事件流：同时支持两种事件模型：捕获型事件和冒泡型事件，但是，捕获型事件先发生。两种事件流会触及DOM中的所有对象，从document对象开始，也在document对象结束。

事件捕获

当你使用事件捕获时，父级元素先触发，子级元素后触发，即div先触发，p后触发。

事件冒泡

当你使用事件冒泡时，子级元素先触发，父级元素后触发，即p先触发，div后触发。

阻止冒泡

• 在W3c中，使用stopPropagation（）方法

• 在IE下设置cancelBubble = true；

在捕获的过程中stopPropagation（）；后，后面的冒泡过程也不会发生了。

阻止捕获

阻止事件的默认行为，例如click <a>后的跳转

• 在W3c中，使用preventDefault（）方法；

• 在IE下设置window.event.returnValue = false;

**42**，介绍事件**“**捕获**”**和**“**冒泡**”**执行顺序和事件的执行次数？

按照W3C标准的事件：首是进入捕获阶段，直到达到目标元素，再进入冒泡阶段

事件执行次数（DOM2-addEventListener）：元素上绑定事件的个数

注意1：前提是事件被确实触发

注意2：事件绑定几次就算几个事件，即使类型和功能完全一样也不会“覆盖”

事件执行顺序：判断的关键是否目标元素

非目标元素：根据W3C的标准执行：捕获->目标元素->冒泡（不依据事件绑定顺序）

目标元素：依据事件绑定顺序：先绑定的事件先执行（不依据捕获冒泡标准）

最终顺序：父元素捕获->目标元素事件1->目标元素事件2->子元素捕获->子元素冒泡->父元素冒泡

注意：子元素事件执行前提 事件确实“落”到子元素布局区域上，而不是简单的具有嵌套关系

在一个DOM上同时绑定两个点击事件：一个用捕获，一个用冒泡。事件会执行几次，先执行冒泡还是捕获？

该DOM上的事件如果被触发，会执行两次（执行次数等于绑定次数）

如果该DOM是目标元素，则按事件绑定顺序执行，不区分冒泡/捕获

如果该DOM是处于事件流中的非目标元素，则先执行捕获，后执行冒泡

**43**，什么是函数节流？介绍一下应用场景和原理？

函数节流(throttle)是指阻止一个函数在很短时间间隔内连续调用。 只有当上一次函数执行后达到规定的时间间隔，才能进行下一次调用。 但要保证一个累计最小调用间隔（否则拖拽类的节流都将无连续效果）

函数节流用于 onresize, onscroll 等短时间内会多次触发的事件

函数节流的原理：使用定时器做时间节流。 当触发一个事件时，先用 setTimout 让这个事件延迟一小段时间再执行。 如果在这个时间间隔内又触发了事件，就 clearTimeout 原来的定时器， 再 setTimeout 一个新的定时器重复以上流程。

函数节流简单实现：

function throttle(method, context) {

     clearTimeout(methor.tId);

     method.tId = setTimeout(function(){

         method.call(context);

     }， 100); // 两次调用至少间隔 100ms

}

// 调用

window.onresize = function(){

    throttle(myFunc, window);

}

**44**，**new** 操作符具体干了什么？

创建实例对象，this 变量引用该对象，同时还继承了构造函数的原型

属性和方法被加入到 this 引用的对象中

新创建的对象由 this 所引用，并且最后隐式的返回 this

**45**，**JavaScript**实现异步编程的方法？

回调函数

事件监听

发布/订阅

Promises对象

Async函数[ES7]

**46**，**web**开发中会话跟踪的方法有哪些

cookie

session

url参数传递

隐藏input

ip地址

**47**，常见的几种数组排序算法**JS**实现

**1**，快速排序

从给定的数据中，随机抽出一项，这项的左边放所有比它小的，右边放比它大的，然后再分别这两边执行上述操作，采用的是递归的思想，总结出来就是　实现一层，分别给两边递归，设置好出口

function fastSort(array,head,tail){

    //考虑到给每个分区操作的时候都是在原有的数组中进行操作的，所以这里head,tail来确定分片的位置

    /\*生成随机项\*/

    var randomnum = Math.floor(ranDom(head,tail));

    var random = array[randomnum];

    /\*将小于random的项放置在其左边  策略就是通过一个临时的数组来储存分好区的结果，再到原数组中替换\*/

    var arrayTemp = [];

    var unshiftHead = 0;

    for(var i = head;i <= tail;i++){

      if(array[i]<random){

        arrayTemp.unshift(array[i]);

        unshiftHead++;

      }else if(array[i]>random){

        arrayTemp.push(array[i]);

      }

      /\*当它等于的时候放哪，这里我想选择放到队列的前面，也就是从unshift后的第一个位置放置\*/

      if(array[i]===random){

        arrayTemp.splice(unshiftHead,0,array[i]);

      }

    }

    /\*将对应项覆盖原来的记录\*/

    for(var j = head , u=0;j <= tail;j++,u++){

      array.splice(j,1,arrayTemp[u]);

    }

    /\*寻找中间项所在的index\*/

    var nowIndex = array.indexOf(random);

    /\*设置出口，当要放进去的片段只有2项的时候就可以收工了\*/

    if(arrayTemp.length <= 2){

      return;

    }

    /\*递归，同时应用其左右两个区域\*/

    fastSort(array,head,nowIndex);

    fastSort(array,nowIndex+1,tail);

}

**2**，插入排序

思想就是在已经排好序的数组中插入到相应的位置，以从小到大排序为例，扫描已经排好序的片段的每一项，如大于，则继续往后，直到他小于一项时，将其插入到这项的前面

function insertSort(array){

    /\*start根据已排列好的项数决定\*/

    var start=1;

    /\*按顺序，每一项检查已排列好的序列\*/

    for(var i=start; i<array.length; start++,i++){

      /\*跟已排好序的序列做对比，并插入到合适的位置\*/

      for(var j=0; j<start; j++){

        /\*小于或者等于时（我们是升序）插入到该项前面\*/

        if(array[i]<=array[j]){

          console.log(array[i]+' '+array[j]);

          array.splice(j,0,array[i]);

          /\*删除原有项\*/

          array.splice(i+1,1);

          break;

        }

      }

    }

}

**3**，冒泡排序

故名思意 ，就是一个个冒泡到最前端或者最后端，主要是通过两两依次比较，以升序为例，如果前一项比后一项大则交换顺序，一直比到最后一对

function bubbleSort(array){

    /\*给每个未确定的位置做循环\*/

    for(var unfix=array.length-1; unfix>0; unfix--){

      /\*给进度做个记录，比到未确定位置\*/

      for(var i=0; i<unfix;i++){

        if(array[i]>array[i+1]){

          var temp = array[i];

          array.splice(i,1,array[i+1]);

          array.splice(i+1,1,temp);

        }

      }

    }

  }

**4**，选择排序

将当前未确定块的min或者max取出来插到最前面或者后面

function selectSort(array){

    /\*给每个插入后的未确定的范围循环，初始是从0开始\*/

    for(var unfixed=0; unfixed<array.length; unfixed++){

      /\*设置当前范围的最小值和其索引\*/

      var min = array[unfixed];

      var minIndex = unfixed;

      /\*在该范围内选出最小值\*/

      for(var j=unfixed+1; j<array.length; j++){

        if(min>array[j]){

          min = array[j];

          minIndex = j;

        }

      }

      /\*将最小值插入到unfixed，并且把它所在的原有项替换成\*/

      array.splice(unfixed,0,min);

      array.splice(minIndex+1,1);

    }

  }

**48**，**svn**与**git**的区别

git是分布式的，svn不是。

git跟svn一样有自己的集中式版本库或服务器。但git更倾向于被使用于分布式模式，克隆版本库后即使没有网络也能够commit文件，查看历史版本记录，创建项目分支等，等网络再次连接上Push到服务器端。

git把内容按元数据方式存储，而svn是按文件。

所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。

git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

git没有一个全局的版本号，svn有。

git的内容完整性优于svn。

因为git的内容存储使用的是SHA-1哈希算法。

git可以有无限个版本库，svn只能有一个指定中央版本库。

当svn中央版本库有问题时，所有工作成员都一起瘫痪直到版本库维修完毕或者新的版本库设立完成。

每一个git都是一个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。如果主要版本库（例如：置於GitHub的版本库）有问题，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

**49**，图片懒加载与预加载

图片懒加载的原理就是暂时不设置图片的src属性，而是将图片的url隐藏起来，比如先写在data-src里面，等某些事件触发的时候(比如滚动到底部，点击加载图片)再将图片真实的url放进src属性里面，从而实现图片的延迟加载

图片预加载是指在一些需要展示大量图片的网站，实现图片的提前加载。从而提升用户体验。常用的方式有两种，一种是隐藏在css的background的url属性里面，一种是通过javascript的Image对象设置实例对象的src属性实现图片的预加载。相关代码如下：

CSS预加载图片方式：

#preload-01 { background: url(http://domain.tld/image-01.png) no-repeat -9999px -9999px; }

#preload-02 { background: url(http://domain.tld/image-02.png) no-repeat -9999px -9999px; }

#preload-03 { background: url(http://domain.tld/image-03.png) no-repeat -9999px -9999px; }

Javascript预加载图片的方式：

function preloadImg(url) {

    var img = new Image();

    img.src = url;

    if(img.complete) {

        //接下来可以使用图片了

        //do something here

    } else {

        img.onload = function() {

            //接下来可以使用图片了

            //do something here

        };

    }

}

**50**，**AMD**和**CMD**的区别

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。

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）。CMD 推崇 as lazy as possible.

CMD 推崇依赖就近，AMD 推崇依赖前置。

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。比如 AMD 里，require 分全局 require 和局部 require，都叫 require。CMD 里，没有全局 require，而是根据模块系统的完备性，提供 seajs.use 来实现模块系统的加载启动。CMD 里，每个 API 都简单纯粹。

**51**，**Less**常用特性

变量(@color = #fff)

混合(Mixin)

内置函数(颜色，字符串，类型判断，数学)

循环

嵌套

运算

导入(@import)

**52**，**ES6**常用特性

变量定义(let和const,可变与不可变，const定义对象的特殊情况)

解构赋值

模板字符串

数组新API(例：Array.from(),entries(),values(),keys())

箭头函数(rest参数，扩展运算符，::绑定this)

Set和Map数据结构(set实例成员值唯一存储key值，map实例存储键值对(key-value))

Promise对象(前端异步解决方案进化史，generator函数，async函数)

Class语法糖(super关键字)

**53**，**src**和**href**的区别

src用于替换当前元素，href用于在当前文档和引用资源之间确立联系。

src是source的缩写，指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部。

Src source,指向外部资源的位置,如果我们添加<script src ="js.js"></script>浏览器会暂停其他资源的下载和处理,直到该资源加载,编译,执行完毕(图片和框架也是如此),这也就是为什么js脚本要放在底部。

src用于替换当前元素,href用于在当前文档和引入资源之间建立联系。

href是Hypertext Reference的缩写，指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，如果我们在文档中添加<link href="common.css" rel="stylesheet"/>那么浏览器会识别该文档为css文件，就会并行下载资源并且不会停止对当前文档的处理。这也是为什么建议使用link方式来加载css，而不是使用@import方式。

Href 超文本链接,指向网络资源所在位置,如果我们在文档添加<link href="common.css" rel="stylesheet"/>浏览器会下载资源并且不会停止对当前文档的处理。

**54.**线程与进程的区别**?**

一个程序至少有一个进程,一个进程至少有一个线程;

多进程拥有独立的内存,多线程共享内存,所以多线程提高了运行效率;

多线程的重要意义在于一个应用程序中,有多个执行程序能够同时执行,但是系统并没有将多线程看成多个独立的应用

**55**，对前端工程师这个职位你是怎么样理解的？

与用户打交道最近的地方,负责用户所能看到的一切的构建,在满足基本功能的同时,添加炫酷的特效,排版等,提升用户体验感,加强产品粘性。

前端是最贴近用户的程序员，前端的能力就是能让产品从 90 分进化到 100 分，甚至更好

参与项目，快速高质量完成实现效果图，精确到 1px；

与团队成员，UI 设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理 hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

**56.**平时如何管理你的项目？

**57**，说说你对 **SVG** 理解**?**

SVG可缩放矢量图形是基于可扩展标记语言 XML，用于描述二维矢量图形的一种图形格式。

SVG 是一种新的二维矢量图形格式，也是规范中的网络矢量图形标准。

SVG 严格遵从 XML 语法，并用文本格式的描述性语言来描述图像内容，因此是一种和图像分辨率无关的矢量图形格式。

特点：

(1)任意放缩   用户可以任意缩放图像显示，而不会破坏图像的清晰度、细节等。

(2)文本独立   SVG图像中的文字独立于图像，文字保留可编辑和可搜寻的状态。也不会再有字体的限制，用户系统即使没有安装某一字体，也会看到和他们制作时完全相同的画面。

(3)较小文件   总体来讲，SVG文件比那些 GIF 和 JPEG 格式的文件要小很多，因而下载也很快。

(4)超强显示效果   SVG图像在屏幕上总是边缘清晰，它的清晰度适合任何屏幕分辨率和打印分辨率。

(5)超级颜色控制   SVG图像提供一个 1600 万种颜色的调色板，支持 ICC 颜色描述文件标准、 RGB 、线 X 填充、渐变和蒙版。

**58**，浅拷贝**vs**深拷贝

拷贝其实就是对象复制，为了解决对象复制是产生的引用类型问题

浅拷贝：利用迭代器，循环对象将对象中的所有可枚举属性复制到另一个对象上，但是浅拷贝的有一个问题就是只是拷贝了对象的一级，其他级还如果是引用类型的值的话依旧解决不了

深拷贝：深拷贝解决了浅拷贝的问题，利用递归的形势便利对象的每一级，实现起来较为复杂，得判断值是数组还是对象；

**59**，区分数组和对象的方法？

1、从原型入手，Array.prototype.isPrototypeOf(obj);

利用isPrototypeOf()方法，判定Array是不是在obj的原型链中，如果是，则返回true,否则false。

Array.prototype.isPrototype([]) //true

2、也可以从构造函数入手，利用对向的constructor属性

3、根据对象的class属性(类属性)，跨原型链调用toString()方法。

Object.prototype.toString.call(Window);

4、Array.isArray()方法。

**60**，说说你对**Promise**的理解

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件监听——更合理和更强大。Promise 有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。但是无法获取到pending状态，在promise中接受两个内置参数分别是resolve（成功）和reject（失败），Promise实例生成以后，可以用then方法分别指定resolved状态和rejected状态的回调函数。then方法可以传递两个回调函数第一个是成功，第二个是失败，失败回调也可以使用promise的catch方法回调，promise还有一个强大的功能那就是all方法可以组合多个promise实例，包装成一个新的 Promise 实例。

**61**，介绍一下**async**和**await**；

async 会将其后的函数（函数表达式或 Lambda）的返回值封装成一个 Promise 对象，而 await 会等待这个 Promise 完成，并将其 resolve 的结果返回出来。

async / await是ES7的重要特性之一，也是目前社区里公认的优秀异步解决方案。目前async / await 在 IE edge中已经可以直接使用了，但是chrome和Node.js还没有支持。幸运的是，babel已经支持async的transform了，所以我们使用的时候引入babel就行。在开始之前我们需要引入以下的package，preset-stage-3里就有我们需要的async/await的编译文件。

**62**，介绍一下**fetch**；

**63**，说说对**es6**的理解（说一下**es6**，知道**es6**吗）

语法糖(箭头函数，类的定义，继承)，以及一些新的扩展（数组，字符串，对象，方法等），对作用域的重新定义，以及异步编程的解决方案（promise，async，await）、解构赋值的出现

**64**，箭头函数的作用域上下文和普通函数作用域上下文的区别

箭头函数其实只是一个密名函数的语法糖，区别在于普通函数作用域中的this有特定的指向，一般指向window，而箭头函数中的this只有一个指向那就是指当前函数所在的对象，其实现原理其实就是类似于之前编程的时候在函数外围定义that一样，用了箭头函数就不用定义that了直接使用this

65.es6如何转为es5？

AJAX篇

1，Ajax 是什么? 如何创建一个Ajax？

ajax的全称：Asynchronous Javascript And XML。异步传输+js+xml。

所谓异步，在这里简单地解释就是：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

            (1)创建XMLHttpRequest对象,也就是创建一个异步调用对象

            (2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息

            (3)设置响应HTTP请求状态变化的函数

            (4)发送HTTP请求

            (5)获取异步调用返回的数据

            (6)使用JavaScript和DOM实现局部刷新

            ajax是一种创建交互式网页的计算

2，同步和异步的区别?

同步的概念应该是来自于OS中关于同步的概念:不同进程为协同完成某项工作而在先后次序上调整(通过阻塞,唤醒等方式).同步强调的是顺序性.谁先谁后.异步则不存在这种顺序性.

同步：浏览器访问服务器请求，用户看得到页面刷新，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,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。

3，如何解决跨域问题?

jsonp、 iframe、window.name、window.postMessage、服务器上设置代理页面

参考：http://qiutc.me/post/cross-domain-collections.html

4，http状态码有那些？分别代表是什么意思？

简单版

             [

                        100  Continue   继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

                        200  OK         正常返回信息

                        201  Created    请求成功并且服务器创建了新的资源

                        202  Accepted   服务器已接受请求，但尚未处理

                        301  Moved Permanently  请求的网页已永久移动到新位置。

                        302 Found       临时性重定向。

                        303 See Other   临时性重定向，且总是使用 GET 请求新的 URI。

                        304  Not Modified 自从上次请求后，请求的网页未修改过。

                        400 Bad Request  服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

                        401 Unauthorized 请求未授权。

                        403 Forbidden   禁止访问。

                        404 Not Found   找不到如何与 URI 相匹配的资源。

                        500 Internal Server Error  最常见的服务器端错误。

                        503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

             ]

5，一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？（流程说的越详细越好）

注：这个问题最主要的是问浏览器的渲染原理

而高手可以根据自己擅长的领域自由发挥，从URL规范、HTTP协议、DNS、CDN、数据库查询、到浏览器流式解析、CSS规则构建、layout、paint、onload/domready、JS执行、JS API绑定等等；

详细版：

            1、浏览器会开启一个线程来处理这个请求，对 URL 分析判断如果是 http 协议就按照 Web 方式来处理;

            2、调用浏览器内核中的对应方法，比如 WebView 中的 loadUrl 方法;

            3、通过DNS解析获取网址的IP地址，设置 UA 等信息发出第二个GET请求;

            4、进行HTTP协议会话，客户端发送报头(请求报头);

            5、进入到web服务器上的 Web Server，如 Apache、Tomcat、Node.JS 等服务器;

            6、进入部署好的后端应用，如 PHP、Java、JavaScript、Python 等，找到对应的请求处理;

            7、处理结束回馈报头，此处如果浏览器访问过，缓存上有对应资源，会与服务器最后修改时间对比，一致则返回304;

            8、浏览器开始下载html文档(响应报头，状态码200)，同时使用缓存;

             9、文档树建立，根据标记请求所需指定MIME类型的文件（比如css、js）,同时设置了cookie;

            10、页面开始渲染DOM，JS根据DOM API操作DOM,执行事件绑定等，页面显示完成。

简洁版：

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

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

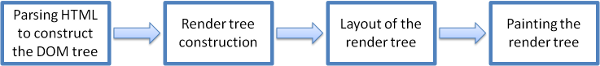
            浏览器对加载到的资源（HTML、JS、CSS等）进行语法解析，建立相应的内部数据结构（如HTML的DOM）；

            载入解析到的资源文件，渲染页面，完成。

浏览器的渲染原理

渲染引擎首先通过网络获得所请求文档的内容，通常以8K分块的方式完成。下面是渲染引擎在取得内容之后的基本流程：

　　解析html以构建dom树 -> 构建render树 -> 布局render树 -> 绘制render树



这里先解释一下几个概念，方便大家理解：

　　DOM Tree：浏览器将HTML解析成树形的数据结构。

　　CSS Rule Tree：浏览器将CSS解析成树形的数据结构。

　　Render Tree: DOM和CSSOM合并后生成Render Tree。

　　layout: 有了Render Tree，浏览器已经能知道网页中有哪些节点、各个节点的CSS定义以及他们的从属关系，从而去计算出每个节点在屏幕中的位置。

　　painting: 按照算出来的规则，通过显卡，把内容画到屏幕上。

　　reflow（回流）：当浏览器发现某个部分发生了点变化影响了布局，需要倒回去重新渲染，内行称这个回退的过程叫 reflow。reflow 会从 <html> 这个 root frame 开始递归往下，依次计算所有的结点几何尺寸和位置。reflow 几乎是无法避免的。现在界面上流行的一些效果，比如树状目录的折叠、展开（实质上是元素的显 示与隐藏）等，都将引起浏览器的 reflow。鼠标滑过、点击……只要这些行为引起了页面上某些元素的占位面积、定位方式、边距等属性的变化，都会引起它内部、周围甚至整个页面的重新渲 染。通常我们都无法预估浏览器到底会 reflow 哪一部分的代码，它们都彼此相互影响着。

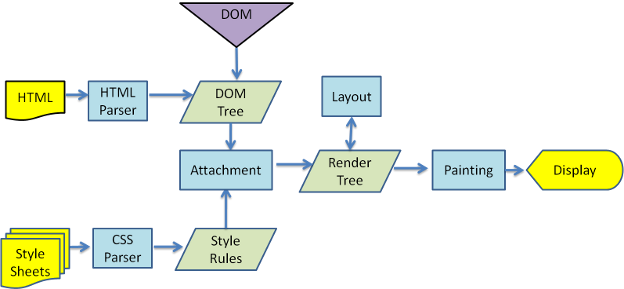
　　repaint（重绘）：改变某个元素的背景色、文字颜色、边框颜色等等不影响它周围或内部布局的属性时，屏幕的一部分要重画，但是元素的几何尺寸没有变。

注意：*(1)display:none* 的节点不会被加入*Render Tree*，而*visibility: hidden* 则会，所以，如果某个节点最开始是不显示的，设为*display:none*是更优的。

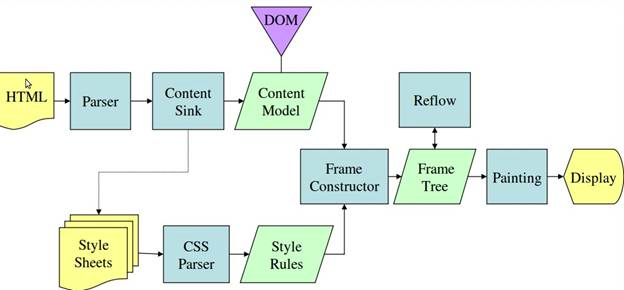
*(2)display:none* 会触发 *reflow*，而 *visibility:hidden* 只会触发 *repaint*，因为没有发现位置变化。

*(3)*有些情况下，比如修改了元素的样式，浏览器并不会立刻*reflow* 或 *repaint* 一次，而是会把这样的操作积攒一批，然后做一次 *reflow*，这又叫异步 *reflow* 或增量异步 *reflow*。但是在有些情况下，比如*resize* 窗口，改变了页面默认的字体等。对于这些操作，浏览器会马上进行 *reflow*。

　　来看看webkit的主要流程：



　　再来看看Geoko的主要流程：



　　Gecko 里把格式化好的可视元素称做“帧树”（Frame tree）。每个元素就是一个帧（frame）。 webkit 则使用”渲染树”这个术语，渲染树由”渲染对象”组成。webkit 里使用”layout”表示元素的布局，Gecko则称为”reflow”。Webkit使用”Attachment”来连接DOM节点与可视化信息以构建渲染树。一个非语义上的小差别是Gecko在HTML与DOM树之间有一个附加的层 ，称作”content sink”，是创建DOM对象的工厂。

　　尽管Webkit与Gecko使用略微不同的术语，这个过程还是基本相同的，如下：

　　1. 浏览器会将HTML解析成一个DOM树，DOM 树的构建过程是一个深度遍历过程：当前节点的所有子节点都构建好后才会去构建当前节点的下一个兄弟节点。

　　2. 将CSS解析成 CSS Rule Tree 。

　　3. 根据DOM树和CSSOM来构造 Rendering Tree。注意：Rendering Tree 渲染树并不等同于 DOM 树，因为一些像Header或display:none的东西就没必要放在渲染树中了。

　　4. 有了Render Tree，浏览器已经能知道网页中有哪些节点、各个节点的CSS定义以及他们的从属关系。下一步操作称之为layout，顾名思义就是计算出每个节点在屏幕中的位置。

　　5. 再下一步就是绘制，即遍历render树，并使用UI后端层绘制每个节点。

　　注意：上述这个过程是逐步完成的，为了更好的用户体验，渲染引擎将会尽可能早的将内容呈现到屏幕上，并不会等到所有的*html*都解析完成之后再去构建和布局*render*树。它是解析完一部分内容就显示一部分内容，同时，可能还在通过网络下载其余内容。

8，请解释 JSONP 的工作原理，以及它为什么不是真正的 AJAX。

jsonp的工作原理是，动态的创建了一个全局方法，并且动态生成script标签，将script标签的src属性变为（接口地址?callback=动态生成方法的方法名）请求数据，而后台则需要将接收到的callback值与数据一同返回，呈现出执行js方法的语句（方法名(数据)），其实就是在请求回来的数据中是执行请求是动态生成的js方法，生成了假象的ajax，所以jsonp只能做get类型请求

9，请解释一下 JavaScript 的同源策略。

同源策略是客户端脚本（尤其是Javascript）的重要的安全度量标准。它最早出自Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。

它的精髓很简单：它认为自任何站点装载的信赖内容是不安全的。当被浏览器半信半疑的脚本运行在沙箱时，它们应该只被允许访问来自同一站点的资源，而不是那些来自其它站点可能怀有恶意的资源。

这里的同源指的是：同协议，同域名和同端口

10，为什么要有同源限制？

我们举例说明：比如一个黑客程序，他利用IFrame把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。

11，创建ajax过程

(1)创建`XMLHttpRequest`对象,也就是创建一个异步调用对象.

(2)创建一个新的`HTTP`请求,并指定该`HTTP`请求的方法、`URL`及验证信息.

(3)设置响应`HTTP`请求状态变化的函数.

(4)发送`HTTP`请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

            var xmlHttp = new XMLHttpRequest();

            xmlHttp.open('GET','demo.php','true');

            xmlHttp.send()

            xmlHttp.onreadystatechange = function(){

                        if(xmlHttp.readyState === 4 & xmlHttp.status === 200){

                        }

            }

11，常见web安全及防护原理

sql注入原理

就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

总的来说有以下几点：

1.永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.永远不要使用动态拼装SQL，可以使用参数化的SQL或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

XSS原理及防范

Xss(cross-site scripting)攻击指的是攻击者往Web页面里插入恶意 html标签或者javascript代码。比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取cookie中的用户私密信息；或者攻击者在论坛中加一个恶意表单当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点。

XSS防范方法

首先代码里对用户输入的地方和变量都需要仔细检查长度和对”<”,”>”,”;”,”’”等字符做过滤；其次任何内容写到页面之前都必须加以encode，避免不小心把html tag 弄出来。这一个层面做好，至少可以堵住超过一半的XSS 攻击。首先，避免直接在cookie 中泄露用户隐私，例如email、密码等等。其次，通过使cookie 和系统ip 绑定来降低cookie 泄露后的危险。这样攻击者得到的cookie 没有实际价值，不可能拿来重放。如果网站不需要再浏览器端对cookie 进行操作，可以在Set-Cookie 末尾加上HttpOnly 来防止javascript 代码直接获取cookie 。尽量采用POST 而非GET 提交表单

12，XSS与CSRF有什么区别吗？

XSS是获取信息，不需要提前知道其他用户页面的代码和数据包。CSRF是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。

要完成一次CSRF攻击，受害者必须依次完成两个步骤：

登录受信任网站A，并在本地生成Cookie。

在不登出A的情况下，访问危险网站B。

CSRF的防御

服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。

通过验证码的方法

13，HTTP和HTTPS

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

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

14，为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性

15，GET和POST的区别，何时使用POST？

GET：一般用于信息获取，使用URL传递参数，对所发送信息的数量也有限制，一般在2000个字符

POST：一般用于修改服务器上的资源，对所发送的信息没有限制。

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值，也就是说Get是通过地址栏来传值，而Post是通过提交表单来传值。

然而，在以下情况中，请使用 POST 请求：

无法使用缓存文件（更新服务器上的文件或数据库）

向服务器发送大量数据（POST 没有数据量限制）

发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

16，ajax的缺点和在IE下的问题？

ajax的缺点

    1、ajax不支持浏览器back按钮。

    2、安全问题 AJAX暴露了与服务器交互的细节。

    3、对搜索引擎的支持比较弱。

    4、破坏了程序的异常机制。

    5、不容易调试。

IE缓存问题

在IE浏览器下，如果请求的方法是GET，并且请求的URL不变，那么这个请求的结果就会被缓存。解决这个问题的办法可以通过实时改变请求的URL，只要URL改变，就不会被缓存，可以通过在URL末尾添加上随机的时间戳参数('t'= + new Date().getTime())

或者：

    open('GET','demo.php?rand=+Math.random()',true);//

    Ajax请求的页面历史记录状态问题

    可以通过锚点来记录状态，location.hash。让浏览器记录Ajax请求时页面状态的变化。

    还可以通过HTML5的history.pushState，来实现浏览器地址栏的无刷新改变

19，前端需要注意哪些SEO

合理的title、description、keywords：搜索对着三项的权重逐个减小，title值强调重点即可，重要关键词出现不要超过2次，而且要靠前，不同页面title要有所不同；description把页面内容高度概括，长度合适，不可   过分堆砌关键词，不同页面description有所不同；keywords列举出重要关键词即可

语义化的HTML代码，符合W3C规范：语义化代码让搜索引擎容易理解网页

重要内容HTML代码放在最前：搜索引擎抓取HTML顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容一定会被抓取

重要内容不要用js输出：爬虫不会执行js获取内容

少用iframe：搜索引擎不会抓取iframe中的内容

非装饰性图片必须加alt

提高网站速度：网站速度是搜索引擎排序的一个重要指标

20.什么是Ajax和JSON，它们的优缺点。

Ajax是异步JavaScript和XML，用于在Web页面中实现异步数据交互。

优点：

    可以使得页面不重载全部内容的情况下加载局部内容，降低数据传输量

    避免用户不断刷新或者跳转页面，提高用户体验

缺点：

    对搜索引擎不友好（

    要实现ajax下的前后退功能成本较大

    可能造成请求数的增加

    跨域问题限制

JSON是一种轻量级的数据交换格式，ECMA的一个子集

优点：轻量级、易于人的阅读和编写，便于机器（JavaScript）解析，支持复合数据类型（数组、对象、字符串、数字）

21，AJAX的请求方式？

GET：GET是http的默认请求方式， 一般用来获取数据。

HEAD：HEAD方法与GET方法一样，都是向服务器发出指定资源的请求。但是，服务器在响应HEAD请求时不会回传资源的内容部分，即：响应主体。这样，我们可以不传输全部内容的情况下，就可以获取服务器的响应头信息。HEAD方法常被用于客户端查看服务器的性能。

POST：POST请求会向指定资源提交数据，请求服务器进行处理。如：表单提交、文件上传。

PUT：PUT请求会身向指定资源位置上传其最新内容，通过该方法客户端可以将指定资源的最新数据传送给服务器取代指定的资源的内容。

DELETE：DELETE请求用于请求服务器删除所请求URI所标识的资源。DELETE请求后指定资源会被删除。

TRACE：返回接受到的请求，用来查看数据经过中间服务器时发生了哪些变动。

OPTIONS：OPTIONS请求与HEAD类似，一般也是用于客户端查看服务器的性能。 这个方法会请求服务器返回该资源所支持的所有HTTP请求方法，该方法会用'\*'来代替资源名称，向服务器发送OPTIONS请求，可以测试服务器功能是否正常。JavaScript的XMLHttpRequest对象进行CORS跨域资源共享时，就是使用OPTIONS方法发送嗅探请求，以判断是否有对指定资源的访问权限。

CONNECT：要求使用SSL和TLS进行TCP通信。

PATCH：请求修改局部数据

**22**，**RESTful**架构

REST是一种架构风格：无状态，以资源为中心，充分利用HTTP协议和URI协议，提供统一的接口定义，使得它作为一种设计Web服务的方法而变得流行。在某种意义上，通过强调URI和HTTP等早期Internet标准，REST是对大型应用程序服务器时代之前的Web方式的回归。

架构约束：

客户-服务器:通信只能由客户端单方面发起，表现为请求-响应的形式。

无状态:通信的会话状态（Session State）应该全部由客户端负责维护。

缓存:响应内容可以在通信链的某处被缓存，以改善网络效率。

统一接口:通信链的组件之间通过统一的接口相互通信，以提高交互的可见性。

分层系统:通过限制组件的行为（即，每个组件只能"看到"与其交互的紧邻层），将架构分解为若干等级的层。

按需代码:支持通过下载并执行一些代码（例如Java Applet、Flash或JavaScript），对客户端的功能进行扩展。

主要特征：

面向资源（Resource Oriented）

可寻址（Addressability）

连通性（Connectedness）

无状态（Statelessness）

统一接口（Uniform Interface）

超文本驱动（Hypertext Driven）

**23**，**cookie** 和**session** 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

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

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

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

5、所以一般情况下将登陆信息等重要信息存放为SESSION其他信息如果需要保留，可以放在COOKIE中

**24**，**Cookie**和**localstorage**、**session storage**的区别

Cookie技术浏览器兼容性好，但操作比较复杂，需要程序员自己封装，源生的Cookie接口不友好,   存储的内容较小， cookie的数据会随着ajax的请求发送到服务端，一般情况主要用在用户登录的时候我们可以通过在 Cookie 中存入一段辨别用户身份的数据，用于后台判断。

WebStorage则不能超过8MB，操作简单；可以代替一些cookie的工作，一般主要是用于存储一些本地数据，购物车数据之类的在安全方面的话，都不安全，一般就是对数据进行一些简单的加密，如base64编码，加密约定之类的东西localstorage、sessionstorage一个是长期存储，一个是会话存储

**25**，**fetch**与**ajax**的区别？

Ajax主要是利用的是XMLHttpRequest对象来请求数据的。

Fetch是window的一个方法 主要特点是

1、第一个参数是URL

2、第二个参数可选参数 可以控制不同的init对象

3、使用了js 中的promise对象

fetch 的第二参数中

1、默认的请求为get请求 可以使用method:post 来进行配置

2、第一步中的 response有许多方法 json() text() formData()

3、Fetch跨域的时候默认不会带cookie ，如果你想在fetch请求里附带cookies之类的凭证信息，可以将 credentials参数设置成 “include” 值。还有就是fetch返回的promise在某些错误的http状态下如400、500等不会reject的错误状态，相反它会被resolve；只有网络错误会导致请求不能完成时，fetch 才会被 reject；所以一般会对fetch请求做一层封装，在resolve中真对于大于200和 小于300的状态返回正确信息，其他则返回错误信息所有的IE浏览器都不会支持 fetch()方法

26，浏览器本地存储

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和localStorage。

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

而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。

27，请你谈谈Cookie的弊端

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。

1，每个特定的域名下最多生成20个cookie

2，IE6或更低版本最多20个cookie

3，IE7和之后的版本最后可以有50个cookie。

4，Firefox最多50个cookie

5，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userdata，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

            1.通过良好的编程，控制保存在cookie中的session对象的大小。

            2.通过加密和安全传输技术（SSL），减少cookie被破解的可能性。

            3.只在cookie中存放不敏感数据，即使被盗也不会有重大损失。

            4.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：

            1.`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.

            2.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。

            3.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。

HTML5篇

**1**，**html5**的新特性

文件类型声明（<!DOCTYPE>）仅有一型：<!DOCTYPE HTML>。

新的解析顺序：不再基于SGML。

绘画 canvas;

用于媒介回放的 video 和 audio 元素;

语意化更好的内容元素**:**article、footer、header、nav、section;

表单控件**:**calendar、date、time、email、url、search;

**input**元素的新类型：date, email, url等。

新的技术**:** webworker, websocket, Geolocation;

新的属性：ping（用于a与area）, charset（用于meta）, async（用于script）。

全域属性：id, tabindex, repeat。

新的全域属性：contenteditable, contextmenu, draggable, dropzone, hidden, spellcheck。

新应用程序接口**:**

HTML Geolocation

HTML Drag and Drop

HTML Local Storage

HTML Application Cache

HTML Web Workers

HTML SSE

HTML Canvas/WebGL

HTML Audio/Video

移除的元素：

纯表现的元素：basefont，big，center，font, s，strike，tt，u;

对可用性产生负面影响的元素：frame，frameset，naframes；

2，什么是Retina 显示屏，带来了什么问题？

retina：一种具备超高像素密度的液晶屏，同样大小的屏幕上显示的像素点由1个变为多个，如在同样带下的屏幕上，苹果设备的retina显示屏中，像素点1个变为4个

在高清显示屏中的位图被放大，图片会变得模糊，因此移动端的视觉稿通常会设计为传统PC的2倍

那么，前端的应对方案是：

设计稿切出来的图片长宽保证为偶数，并使用backgroud-size把图片缩小为原来的1/2

//例如图片宽高为：200px\*200px，那么写法如下

.css{width:100px;height:100px;background-size:100px 100px;}

其它元素的取值为原来的1/2，例如视觉稿40px的字体，使用样式的写法为20px

.css{font-size:20px}

3，ios系统中元素被触摸时产生的半透明灰色遮罩怎么去掉？

ios用户点击一个链接，会出现一个半透明灰色遮罩, 如果想要禁用，可设置-webkit-tap-highlight-color的alpha值为0，也就是属性值的最后一位设置为0就可以去除半透明灰色遮罩

a,button,input,textarea{-webkit-tap-highlight-color: rgba(0,0,0,0;)}

4，部分android系统中元素被点击时产生的边框怎么去掉

android用户点击一个链接，会出现一个边框或者半透明灰色遮罩, 不同生产商定义出来额效果不一样，可设置-webkit-tap-highlight-color的alpha值为0去除部分机器自带的效果

            a,button,input,textarea{

                        -webkit-tap-highlight-color: rgba(0,0,0,0;)

                        -webkit-user-modify:read-write-plaintext-only;

            }

            -webkit-user-modify有个副作用，就是输入法不再能够输入多个字符

            另外，有些机型去除不了，如小米2

            对于按钮类还有个办法，不使用a或者input标签，直接用div标签

5，webkit表单元素的默认外观怎么重置

通用

.css{-webkit-appearance:none;}

伪元素改变number类型input框的默认样式

                        input[type=number]::-webkit-textfield-decoration-container {

                                    background-color: transparent;

                        }

                        input[type=number]::-webkit-inner-spin-button {

                                     -webkit-appearance: none;

                        }

                        input[type=number]::-webkit-outer-spin-button {

                                    -webkit-appearance: none;

                        }

6，webkit表单输入框placeholder的颜色值能改变么

input::-webkit-input-placeholder{color:#AAAAAA;}

input:focus::-webkit-input-placeholder{color:#EEEEEE;}

7，禁止ios 长按时不触发系统的菜单，禁止ios&android长按时下载图片

.css{-webkit-touch-callout: none}

8，禁止ios和android用户选中文字

.css{-webkit-user-select:none}

9，打电话发短信写邮件怎么实现

打电话：            <a href="tel:0755-10086">打电话给:0755-10086</a>

发短信，winphone系统无效         <a href="sms:10086">发短信给: 10086</a>

写邮件：            <a href=“mailto:peun@foxmail.com">peun@foxmail.com</a>

10，audio元素和video元素在ios和andriod中无法自动播放

应对方案：触屏即播

$('html').one('touchstart',function(){

    audio.play()

})

11，闪屏怎么解决

通过transform的3d属性改去硬件加速可以解决闪屏问题

开启硬件加速

1，解决页面闪白

2，保证动画流畅

            .css {

                        -webkit-transform: translate3d(0, 0, 0);

                        -moz-transform: translate3d(0, 0, 0);

                        -ms-transform: translate3d(0, 0, 0);

                        transform: translate3d(0, 0, 0);

            }

12，取消input在ios下，输入的时候英文首字母的默认大写

<input autocapitalize="off" autocorrect="off" />

13，android 上去掉语音输入按钮

input::-webkit-input-speech-button {display: none}

14，设计高性能CSS3动画的几个要素

尽可能地使用合成属性transform和opacity来设计CSS3动画

不使用position的left和top来定位

利用translate3D开启GPU加速

15，fixed bug

ios下fixed元素容易定位出错，软键盘弹出时，影响fixed元素定位

android下fixed表现要比iOS更好，软键盘弹出时，不会影响fixed元素定位

ios4下不支持position:fixed

解决方案

可用isroll.js，暂无完美方案

16，移动H5前端性能优化指南

            1. PC优化手段在Mobile侧同样适用

            2. 在Mobile侧我们提出三秒种渲染完成首屏指标

            3. 基于第二点，首屏加载3秒完成或使用Loading

            4. 基于联通3G网络平均338KB/s(2.71Mb/s)，所以首屏资源不应超过1014KB

            5. Mobile侧因手机配置原因，除加载外渲染速度也是优化重点

            6. 基于第五点，要合理处理代码减少渲染损耗

            7. 基于第二、第五点，所有影响首屏加载和渲染的代码应在处理逻辑中后置

            8. 加载完成后用户交互使用时也需注意性能

            优化指南

            [加载优化]

            加载过程是最为耗时的过程，可能会占到总耗时的80%时间，因此是优化的重点

            · 减少HTTP请求

            因为手机浏览器同时响应请求为4个请求（Android支持4个，iOS 5后可支持6个），所以要尽量减少页面的请求数，首次加载同时请求数不能超过4个

            a) 合并CSS、JavaScript

            b) 合并小图片，使用雪碧图

            · 缓存

            使用缓存可以减少向服务器的请求数，节省加载时间，所以所有静态资源都要在服务器端设置缓存，并且尽量使用长Cache（长Cache资源的更新可使用时间戳）

            a) 缓存一切可缓存的资源

            b) 使用长Cache（使用时间戳更新Cache）

            c) 使用外联式引用CSS、JavaScript

            · 压缩HTML、CSS、JavaScript

            减少资源大小可以加快网页显示速度，所以要对HTML、CSS、JavaScript等进行代码压缩，并在服务器端设置GZip

            a) 压缩（例如，多余的空格、换行符和缩进）

            b) 启用GZip

            · 无阻塞

            写在HTML头部的JavaScript（无异步），和写在HTML标签中的Style会阻塞页面的渲染，因此CSS放在页面头部并使用Link方式引入，避免在HTML标签中写Style，JavaScript放在页面尾

            部或使用异步方式加载

            · 使用首屏加载

            首屏的快速显示，可以大大提升用户对页面速度的感知，因此应尽量针对首屏的快速显示做优化

            · 按需加载

            将不影响首屏的资源和当前屏幕资源不用的资源放到用户需要时才加载，可以大大提升重要资源的显示速度和降低总体流量

            PS：按需加载会导致大量重绘，影响渲染性能

            a) LazyLoad

            b) 滚屏加载

            c) 通过Media Query加载

            · 预加载

            大型重资源页面（如游戏）可使用增加Loading的方法，资源加载完成后再显示页面。但Loading时间过长，会造成用户流失

            对用户行为分析，可以在当前页加载下一页资源，提升速度

            a) 可感知Loading(如进入空间游戏的Loading)

            b) 不可感知的Loading（如提前加载下一页）

            · 压缩图片

            图片是最占流量的资源，因此尽量避免使用他，使用时选择最合适的格式（实现需求的前提下，以大小判断），合适的大小，然后使用智图压缩，同时在代码中用Srcset来按需显示

            PS：过度压缩图片大小影响图片显示效果

            a) 使用智图（ http://zhitu.isux.us/ ）

            b) 使用其它方式代替图片(1. 使用CSS3 2. 使用SVG 3. 使用IconFont)

            c) 使用Srcset

            d) 选择合适的图片(1. webP优于JPG 2. PNG8优于GIF)

            e) 选择合适的大小（1. 首次加载不大于1014KB 2. 不宽于640（基于手机屏幕一般宽度））

            · 减少Cookie

            Cookie会影响加载速度，所以静态资源域名不使用Cookie

            · 避免重定向

            重定向会影响加载速度，所以在服务器正确设置避免重定向

            · 异步加载第三方资源

            第三方资源不可控会影响页面的加载和显示，因此要异步加载第三方资源

            [脚本执行优化]

            脚本处理不当会阻塞页面加载、渲染，因此在使用时需当注意

            · CSS写在头部，JavaScript写在尾部或异步

            · 避免图片和iFrame等的空Src

            空Src会重新加载当前页面，影响速度和效率

            · 尽量避免重设图片大小

            重设图片大小是指在页面、CSS、JavaScript等中多次重置图片大小，多次重设图片大小会引发图片的多次重绘，影响性能

            · 图片尽量避免使用DataURL

            DataURL图片没有使用图片的压缩算法文件会变大，并且要解码后再渲染，加载慢耗时长

            [CSS优化]

            · 尽量避免写在HTML标签中写Style属性

            · 避免CSS表达式

                        CSS表达式的执行需跳出CSS树的渲染，因此请避免CSS表达式

            · 移除空的CSS规则

                        空的CSS规则增加了CSS文件的大小，且影响CSS树的执行，所以需移除空的CSS规则

            · 正确使用Display的属性

                        Display属性会影响页面的渲染，因此请合理使用

                        a) display:inline后不应该再使用width、height、margin、padding以及float

                        b) display:inline-block后不应该再使用float

                        c) display:block后不应该再使用vertical-align

                        d) display:table-\*后不应该再使用margin或者float

            · 不滥用Float

                        Float在渲染时计算量比较大，尽量减少使用

             · 不滥用Web字体

                        Web字体需要下载，解析，重绘当前页面，尽量减少使用

            · 不声明过多的Font-size

                        过多的Font-size引发CSS树的效率

            · 值为0时不需要任何单位

                        为了浏览器的兼容性和性能，值为0时不要带单位

            · 标准化各种浏览器前缀

                        a) 无前缀应放在最后

                        b) CSS动画只用 （-webkit- 无前缀）两种即可

                        c) 其它前缀为 -webkit- -moz- -ms- 无前缀 四种，（-o-Opera浏览器改用blink内核，所以淘汰）

            · 避免让选择符看起来像正则表达式 高级选择器执行耗时长且不易读懂，避免使用

            [JavaScript执行优化]

            · 减少重绘和回流

                        a) 避免不必要的Dom操作

                        b) 尽量改变Class而不是Style，使用classList代替className

                        c) 避免使用document.write

                        d) 减少drawImage

            · 缓存Dom选择与计算

                        每次Dom选择都要计算，缓存他

            · 缓存列表.length

                        每次.length都要计算，用一个变量保存这个值

            · 尽量使用事件代理，避免批量绑定事件

            · 尽量使用ID选择器

                        ID选择器是最快的

            · TOUCH事件优化

                        使用touchstart、touchend代替click，因快影响速度快。但应注意Touch响应过快，易引发误操作

            [渲染优化]

            · HTML使用Viewport

                        Viewport可以加速页面的渲染，请使用以下代码

                        <meta name=”viewport” content=”width=device-width, initial-scale=1″>

            · 减少Dom节点

                        Dom节点太多影响页面的渲染，应尽量减少Dom节点

            · 动画优化

                        a) 尽量使用CSS3动画

                        b) 合理使用requestAnimationFrame动画代替setTimeout

                        c) 适当使用Canvas动画 5个元素以内使用css动画，5个以上使用Canvas动画（iOS8可使用webGL）

            · 高频事件优化

                        Touchmove、Scroll 事件可导致多次渲染

                        a) 使用requestAnimationFrame监听帧变化，使得在正确的时间进行渲染

                        b) 增加响应变化的时间间隔，减少重绘次数

            · GPU加速

                        CSS中以下属性（CSS3 transitions、CSS3 3D transforms、Opacity、Canvas、WebGL、Video）来触发GPU渲染，请合理使用（PS：过渡使用会引发手机过耗电增加）

17，CSS动画与Canvas动画性能优劣分析

a、CSS动画更为流畅、但内存占用过高，动画元素在5个以内更为推荐；

b、Canvas动画存在丢帧现象，这一现象在android中低端手机中表现更为明显；

c、5个以内动画元素，选用CSS动画，80%的设备帧频可达80以上。

18、移动端怎么自适应？

移动自适应的话一般我用的话就是rem，通过rem单位配合媒体查询来完成几个屏幕的适配，以iphone6位标准，向上适配plus，向下适配5，不做过多的适配，有时候有要求的话也用js来控制rem的大小达到适配效果，rem是相对单位，相对html元素的font-size值的属性，比如font-size:100px; 的话1rem就得100px，这样我子需要通过改变html的font-size就可以达到适配效果

19、如何开启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

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;

通过**-webkit-transform:transition3d/translateZ**开启**GPU**硬件加速的适用范围：

使用很多大尺寸图片(尤其是PNG24图)进行动画的页面。

页面有很多大尺寸图片并且进行了css缩放处理，页面可以滚动时。

使用background-size:cover设置大尺寸背景图，并且页面可以滚动时

编写大量DOM元素进行CSS3动画时(transition/transform/keyframes/absTop&Left)

使用很多PNG图片拼接成CSS Sprite时

20、Svg和canvas的区别

canvas是html5提供的新元素<canvas>，而svg存在的历史要比canvas久远，已经有十几年了。svg并不是html5专有的标签，最初svg是用xml技术（超文本扩展语言，可以自定义标签或属性）描述二维图形的语言。在H5中看似canvas与svg很像，但是，他们有巨大的差别。

  首先，从它们的功能上来讲，canvas可以看做是一个画布。，其绘制出来的图形为标量图，因此，可以在canvas中引入jpg或png这类格式的图片，在实际开发中，大型的网络游戏都是用canvas画布做出来的，并且canvas的技术现在已经相当的成熟。另外，我们喜欢用canvas来做一些统计用的图表，如柱状图曲线图或饼状图等。而svg，所绘制的图形为矢量图，所以其用法上受到了限制。因为只能绘制矢量图，所以svg中不能引入普通的图片，因为矢量图的不会失真的效果，在项目中我们会用来做一些动态的小图标。但是由于其本质为矢量图，可以被无限放大而不会失真，这很适合被用来做地图，而百度地图就是用svg技术做出来的。

另外从技术发面来讲canvas里面绘制的图形不能被引擎抓取，如我们要让canvas里面的一个图片跟随鼠标事件：canvas.onmouseover=function(){}。而svg里面的图形可以被引擎抓取，支持事件的绑定。另外canvas中我们绘制图形通常是通过javascript来实现，svg更多的是通过标签来来实现，如在svg中绘制正矩形形就要用<rect>，这里我们不能用属性style="width:XXX;height:XXX;”来定义

SVG

　　　　SVG 是一种使用 XML 描述 2D 图形的语言。

　　　　SVG 基于 XML，这意味着 SVG DOM 中的每个元素都是可用的。您可以为某个元素附加 JavaScript 事件处理器。

　　　　在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

　　　　特点：

　　　　    　　不依赖分辨率

　　　　    　　支持事件处理器

　　　　    　　最适合带有大型渲染区域的应用程序（比如谷歌地图）

　　　　　　    复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）

　　　　    　　不适合游戏应用

Canvas

　　　　Canvas 通过 JavaScript 来绘制 2D 图形。

　　　　Canvas 是逐像素进行渲染的。

　　　　在 canvas 中，一旦图形被绘制完成，它就不会继续得到浏览器的关注。如果其位置发生变化，那么整个场景也需要重新绘制，包括任何或许已被图形覆盖的对象。

　　　　特点：

　　　    　　依赖分辨率

　　　　    　不支持事件处理器

　　　　　　 弱的文本渲染能力

　　　　　　 能够以 .png 或 .jpg 格式保存结果图像

　　　　    　最适合图像密集型的游戏，其中的许多对象会被频繁重绘

21，物理**1**像素怎么实现

可以使用after插入的形式将宽度设置为200%，然后通过css transfrom 的scale 缩放0.5倍；

**22**，响应式布局的原理

Meta标签定义

使用 viewport meta 标签在手机浏览器上控制布局

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

通过快捷方式打开时全屏显示

<meta name="apple-mobile-web-app-capable" content="yes" />

隐藏状态栏

<meta name="apple-mobile-web-app-status-bar-style" content="blank" />

iPhone会将看起来像电话号码的数字添加电话连接，应当关闭

<meta name="format-detection" content="telephone=no" />

使用Media Queries适配对应样式

常用于布局的CSS Media Queries有以下几种：

设备类型(media type):

all所有设备；

screen 电脑显示器；

print打印用纸或打印预览视图；

handheld便携设备；

tv电视机类型的设备；

speech语意和音频盒成器；

braille盲人用点字法触觉回馈设备；

embossed盲文打印机；

projection各种投影设备；

tty使用固定密度字母栅格的媒介，比如电传打字机和终端。

设备特性(media feature):

width浏览器宽度；

height浏览器高度；

device-width设备屏幕分辨率的宽度值；

device-height设备屏幕分辨率的高度值；

orientation浏览器窗口的方向纵向还是横向，当窗口的高度值大于等于宽度时该特性值为portrait，否则为landscape；

aspect-ratio比例值，浏览器的纵横比；

device-aspect-ratio比例值，屏幕的纵横比。

设置多种视图宽度

@media only screen and (min-width:768px)and(max-width:1024px){}

@media only screen and (width:320px)and (width:768px){}

百分比布局

宽度不固定，可以使用百分比

#head{width:100%;}

#content{width:50%;}

响应式图片

#wrap img{

  max-width:100%;

  height:auto;

}

字体设置

一个响应式的字体应关联它的父容器的宽度，这样才能适应客户端屏幕。css3引入了新的单位叫做rem,和em类似但对于Html元素，rem更方便使用。em是相对于根元素的，需重置根元素字体大小：

html{font-size:100%;}

完成后，可以定义响应式字体：

@media (min-width:640px){body{font-size:1rem;}}

@media (min-width:960px){body{font-size:1.2rem;}}

@media (min-width:1200px){body{font-size:2rem;}}

**23**，常见**dpr**

ip6 2、ip6+ 3、nexus5 3

**24**，移动端适配

1.自适应方式

2.viewport方式

3.rem方式

4.scale()方式缩放

**25**，常用**meta**

IE相关

避免IE使用兼容模式 <meta http-equiv="X-UA-Compatible" content="IE=edge" />

为不支持viewport的IE Mobile设定宽度 <meta name="MobileOptimized" content="320" />

是否对手持设备友好 <meta name="HandheldFriendly" content="true" />

UC浏览器

强制竖屏 <meta name="screen-orientation" content="portrait" />

强制全屏 <meta name="full-screen" content="yes" />

应用模式 <meta name="browsermode" content="application" />

X5浏览器

强制竖屏 <meta name="x5-orientation" content="portrait" />

强制全屏 <meta name="x5-fullscreen" content="yes" />

应用模式 <meta name="x5-page-mode" content="app" />

360

开启极速模式 <meta name="renderer" content="webkit" />

iphone相关

启动webapp功能 删除工具栏和菜单栏 <meta name="apple-mobile-web-app-capable" content="yes">

控制状态栏颜色 <meta name="apple-mobile-web-app-status-bar-style" content="black" />

开启号码或邮箱检测 <meta name="format-detection" content="telephone=no" />

添加主屏幕icon <link rel="apple-touch-icon" href="touch-icon-iphone.png”>

**26**，**html5**的新**api**

重力感应 DeviceOrientation

手机加速传感器 DeviceMotion

history 详细

以Pjax方式实现SPA history.pushState history.replaceState 解决ajax不能后退前进问题

储存

客户端储存：localStorage sessionStorage>只存在一个会话周期内

跨域

postMessage

27，你们怎么嵌套h5页面的

一般都是我前端这边开发好h5页面给安卓和ios一个地址，然后告诉他们我有哪些方法，然后在做一些安卓ios的接口调用，安卓和ios会吧他们的方法提供一个接口给我，我直接调用就好，一般都是调用一些分享，跳转之类的方法

28，webapp你主要做的是哪个模块，是页面布局做的多还是交互做的多

webapp部分基本都是我自己负责开发，页面交互都做，因为当时刚去公司的时候公司也刚往这个方向发展，让我也一直就负责移动业务这块的内容

29，bootstrap用吗？

用，bootstrap组要就是用一些他的样式组件，还有最强大的就是他的删格化系统了，做响应式的网站，我们公司的官网就是用的bootstrap做的，后台管理系统也是，当时也是我负责给我们公司的后台分享如何使用bootstrap的

Angular篇

1，ng-if 跟ng-show/hide 的区别有哪些？

第一点区别是，ng-if 在后面表达式为 true 的时候才创建这个 dom 节点，ng-show 是初始时就创建了，用 display:block 和 display:none 来控制显示和不显示。

第二点区别是，ng-if 会（隐式地）产生新作用域，ng-switch 、 ng-include 等会动态创建一块界面的也是如此。

这样会导致，在 ng-if 中用基本变量绑定 ng-model，并在外层 div 中把此 model 绑定给另一个显示区域，内层改变时，外层不会同步改变，因为此时已经是两个变量了。

<p>{{name}}</p>

<div ng-if="true">

<input type="text" ng-model="name">

</div>

ng-show 不存在此问题，因为它不自带一级作用域。

避免这类问题出现的办法是，始终将页面中的元素绑定到对象的属性（data.x）而不是直接绑定到基本变量（x）上。

2，ng-repeat迭代数组的时候，如果数组中有相同值，会有什么问题，如何解决？

会提示 Duplicates in a repeater are not allowed. 加 track by $index 可解决。当然，也可以 trace by 任何一个普通的值，只要能唯一性标识数组中的每一项即可（建立 dom 和数据之间的关联）。

3，ng-click 中写的表达式，能使用 JS 原生对象上的方法吗？

不止是 ng-click 中的表达式，只要是在页面中，都不能直接调用原生的 JS 方法，因为这些并不存在于与页面对应的 Controller 的 $scope 中。

4，{{now | 'yyyy-MM-dd'}} 这种表达式里面，竖线和后面的参数通过什么方式可以自定义？

filter，格式化数据，接收一个输入，按某规则处理，返回处理结果。

ng 内置的 filter 有九种：

                  date（日期）

                  currency（货币）

                  limitTo（限制数组或字符串长度）

                  orderBy（排序）

                  lowercase（小写）

                  uppercase（大写）

                  number（格式化数字，加上千位分隔符，并接收参数限定小数点位数）

                  filter（处理一个数组，过滤出含有某个子串的元素）

                  json（格式化 json 对象）

filter 有两种使用方法，一种是直接在页面里：

<p>{{now | date : 'yyyy-MM-dd'}}</p>

另一种是在 js 里面用：

// $filter('过滤器名称')(需要过滤的对象, 参数1, 参数2,...)

$filter('date')(now, 'yyyy-MM-dd hh:mm:ss’);

自定义 filter

// 形式

app.filter('过滤器名称',function(){

    return function(需要过滤的对象,过滤器参数1,过滤器参数2,...){

        //...做一些事情

        return 处理后的对象;

    }

});

// 栗子

app.filter('timesFilter', function(){

    return function(item, times){

        var result = '';

        for(var i = 0; i < times; i++){

            result += item;

        }

        return result;

    }

})

5，factory、service 和provider 是什么关系？

factory：把 service 的方法和数据放在一个对象里，并返回这个对象

service：通过构造函数方式创建 service，返回一个实例化对象

provider：创建一个可通过 config 配置的 service，$get 中返回的，就是用 factory 创建 service 的内容

从底层实现上来看，service 调用了 factory，返回其实例；factory 调用了 provider，返回其 $get 中定义的内容。factory 和 service 功能类似，只不过 factory 是普通 function，可以返回任何东西（return 的都可以被访问，所以那些私有变量怎么写，你懂的）；service 是构造器，可以不返回（绑定到 this 的都可以被访问）；provider 是加强版 factory，返回一个可配置的 factory。

6，angular 的数据绑定采用什么机制？详述原理

使用的脏检查机制，所谓的双向绑定，其实就是从界面的操作能实时反映到数据，数据的变更能实时展现到界面。AngularJS在$scope变量中使用脏值检查来实现了数据双向绑定，并且可以通过$scope.$watch来监听变化触发回调；

angular中使用的是脏检查机制，在angular中每次你绑定一些东西到你的UI上时你就会往$watch队列里插入一条$watch，当我们的模版加载完毕时，也就是在linking阶段（Angular分为compile阶段和linking阶段---译者注），Angular解释器会寻找每个directive，然后生成每个需要的$watch。当浏览器接受到可以被angular context处理的事件时就会触发digest循环，这个循环是由两个更小的循环组合起来的，一个是$watch列表，一个是$evalAsync列表，而$watch列表在$digest循环中被“脏值检查”解析，在digest将会遍历我们的watch，然后询问它是否有属性和值的变化，直到$watch队列都检查过，在检查数据变化的时候，由于并不知道这个事件是对哪些数据进行了更改，以及这个事件有可能造成事件之外的其他任何地方的数据更改，所以必须进行一次大检查，将所有“注册”过的值全部检查一遍，一次检查称为一个周期，每次最少检查两遍，因为第二遍用来确认，前一遍的变动中是否有数据的变动，导致了其他数据的变动，如果第二次有变动的话，会再执行一遍，直到最后两次完全一致，则停止检查（其实就是个（递归（遍历））的过程），考虑到内存的消耗和死循环的风险，脏检查每个周期最多递归执行10遍，如果超过10遍就会抛出一个错误。当$digest循环结束时，DOM相应地变化。

在angular中

ng-click，ng-change，ng-blur...就是对各类用户事件的封装

$timeout，$http，$window，$location...就是对各种JS/API事件的封装

ng-model，以及控制器中的数据，就是对值的“注册”

$scope 本质是一个总的事件逻辑的封装容器，同时抽象为数据载体，实质上数据都存在于浏览器堆内存中

$scope.apply() & $scope.digest() 即Angular中的“数据大检查”的function

所以如果我们使用了非Angular封装的事件改编数据时，要手动执行一次大检查

由于Angular这种脏检查的方法效率不高，如果一个页面绑定的view超过2000个，就可能存在比较明显的性能问题，官方称之为“脏检查”

举个例子

<button ng-click="val=val+1">increase 1</button>

click 时会产生一次更新的操作（至少触发两次 $digest 循环）

按下按钮

浏览器接收到一个事件，进入到angular context

$digest 循环开始执行，查询每个 $watch 是否变化

由于监视$scope.val 的 $watch 报告了变化，因此强制再执行一次 $digest 循环

新的 $digest 循环未检测到变化

浏览器拿回控制器，更新 $scope.val 新值对应的 dom

$digest 循环的上限是 10 次（超过 10次后抛出一个异常，防止无限循环）。

7，angular 中控制器之间如何通信？

1、Service

2、events,指定绑定的事件

3、使用 $rootScope

4、controller之间直接使用$parent, $$childHead等

5、directive 指定属性进行数据绑定

8，一个angular 应用应当如何良好地分层？

目录结构的划分

对于小型项目，可以按照文件类型组织，比如：

css

js

  controllers

  models

  services

  filters

templates

但是对于规模较大的项目，最好按业务模块划分，比如：

css

modules

  account

    controllers

    models

    services

    filters

    templates

  disk

    controllers

    models

    services

    filters

    templates

modules 下最好再有一个 common 目录来存放公共的东西。

逻辑代码的拆分

作为一个 MVVM 框架，Angular 应用本身就应该按照 模型，视图模型（控制器），视图来划分。

这里逻辑代码的拆分，主要是指尽量让 controller 这一层很薄。提取共用的逻辑到 service 中 （比如后台数据的请求，数据的共享和缓存，基于事件的模块间通信等），提取共用的界面操作到 directive 中（比如将日期选择、分页等封装成组件等），提取共用的格式化操作到 filter 中等等。

在复杂的应用中，也可以为实体建立对应的构造函数，比如硬盘（Disk）模块，可能有列表、新建、详情这样几个视图，并分别对应的有 controller，那么可以建一个 Disk 构造函数，里面完成数据的增删改查和验证操作，有跟 Disk 相关的 controller，就注入 Disk 构造器并生成一个实例，这个实例就具备了增删改查和验证方法。这样既层次分明，又实现了复用（让 controller 层更薄了）。

9，angular 应用常用哪些路由库，各自的区别是什么？

Angular1.x 中常用 ngRoute 和 ui.router，还有一种为 Angular2 设计的 new router（面向组件）。后面那个没在实际项目中用过，就不讲了。

无论是 ngRoute 还是 ui.router，作为框架额外的附加功能，都必须以 模块依赖 的形式被引入。

区别

ngRoute 模块是 Angular 自带的路由模块，而 ui.router 模块是基于 ngRoute模块开发的第三方模块。

ui.router 是基于 state （状态）的， ngRoute 是基于 url 的，ui.router模块具有更强大的功能，主要体现在视图的嵌套方面。

使用 ui.router 能够定义有明确父子关系的路由，并通过 ui-view 指令将子路由模版插入到父路由模板的 <div ui-view></div>中去，从而实现视图嵌套。而在 ngRoute 中不能这样定义，如果同时在父子视图中 使用了 <div ng-view></div>会陷入死循环。

分属不同团队进行开发的 angular 应用，如果要做整合，可能会遇到哪些问题，如何解决？

可能会遇到不同模块之间的冲突。

比如一个团队所有的开发在 moduleA 下进行，另一团队开发的代码在 moduleB 下

angular.module('myApp.moduleA', [])

    .factory('serviceA', function(){

        ...

    })

angular.module('myApp.moduleB', [])

    .factory('serviceA', function(){

        ...

    })

angular.module('myApp', ['myApp.moduleA', 'myApp.moduleB'])

会导致两个 module 下面的 serviceA 发生了覆盖。

貌似在 Angular1.x 中并没有很好的解决办法，所以最好在前期进行统一规划，做好约定，严格按照约定开发，每个开发人员只写特定区块代码。

10，angular 的缺点有哪些？

强约束

导致学习成本较高，对前端不友好。

但遵守 AngularJS 的约定时，生产力会很高，对 Java 程序员友好。

不利于 SEO

因为所有内容都是动态获取并渲染生成的，搜索引擎没法爬取。

一种解决办法是，对于正常用户的访问，服务器响应 AngularJS 应用的内容；对于搜索引擎的访问，则响应专门针对 SEO 的HTML页面。

性能问题

作为 MVVM 框架，因为实现了数据的双向绑定，对于大数组、复杂对象会存在性能问题。

可以用来 优化 Angular 应用的性能 的办法：

减少监控项（比如对不会变化的数据采用单向绑定）

主动设置索引（指定 track by，简单类型默认用自身当索引，对象默认使用 $$hashKey，比如改为 track by item.id）

降低渲染数据量（比如分页，或者每次取一小部分数据，根据需要再取）

数据扁平化（比如对于树状结构，使用扁平化结构，构建一个 map 和树状数据，对树操作时，由于跟扁平数据同一引用，树状数据变更会同步到原始的扁平数据）

另外，对于Angular1.x ，存在 脏检查 和 模块机制 的问题。

移动端

可尝试 Ionic，但并不完善。

11，详述angular 的“依赖注入”？

依赖注入是一种软件设计模式，目的是处理代码之间的依赖关系，减少组件间的耦合。

 参考：https://segmentfault.com/a/1190000005836443

12，解释下什么是$rootScrope以及和$scope的区别？

通俗的说$rootScrope 页面所有$scope的父亲

如何产生$rootScope和$scope吧。

step1:Angular解析ng-app然后在内存中创建$rootScope。

step2:angular回继续解析，找到{{}}表达式，并解析成变量。

step3:接着会解析带有ng-controller的div然后指向到某个controller函数。这个时候在这个controller函数变成一个$scope对象实例。

13，如何取消$timeout, 以及停止一个$watch()?

停止 $timeout我们可以用cancel：

var customTimeout = $timeout(function () {

 // your code

}, 1000);

$timeout.cancel(customTimeout);

停掉一个$watch：

// .$watch() 会返回一个停止注册的函数

function that we store to a variable

var deregisterWatchFn = $rootScope.$watch(‘someGloballyAvailableProperty', function (newVal) {

 if (newVal) {

  // we invoke that deregistration function, to disable the watch

  deregisterWatchFn();

  ...

 }

});

14，Angular Directive中restrict 中分别可以怎样设置？scope中@,=,&有什么区别？

restrict中可以分别设置:

     A匹配属性

     E匹配标签

     C匹配class

     M 匹配注释

当然你可以设置多个值比如AEC,进行多个匹配。

在scope中，@,=,&在进行值绑定时分别表示

     @获取一个设置的字符串，它可以自己设置的也可以使用{{yourModel}}进行绑定的;

     = 双向绑定，绑定scope上的一些属性；

&用于执行父级scope上的一些表达式，常见我们设置一些需要执行的函数

15，$apply()和$digest()的区别

安全性：$apply()可以接收一个参数作为function()，这个 function 会被包装到一个 try … catch 块中，所以一旦有异常发生，该异常会被 $exceptionHandler service 处理。

$apply会使ng进入 $digest cycle , 并从$rootScope开始遍历(深度优先)检查数据变更。

$digest仅会检查该scope和它的子scope，当你确定当前操作仅影响它们时，用$digest可以稍微提升性能。

16，compile和link的区别：

性能力（性能和能力）

编译的时候，compile转换dom，碰到绑定监听器的地方就先存着，有几个存几个，到最后汇总成一个link函数，一并执行，提升了性能。

function compile(tElement, tAttrs, transclude) { ... }tElement为编译前的element

function link(scope, iElement, iAttrs, controller) { ... }  iElement为编译后的element，已经与作用域关联起来，所以可以数据绑定

如果指令只进行DOM的修改，不进行数据绑定，那么配置在compile函数中，如果指令要进行数据绑定，那么配置在link函数中。

17，两个平级界面块 **a** 和 **b**，如果 **a** 中触发一个事件，有哪些方式能让 **b** 知道？详述原理

这个问题换一种说法就是，如何在平级界面模块间进行通信。有两种方法，一种是共用服务，一种是基于事件。

共用服务

在 Angular 中，通过 factory 可以生成一个单例对象，在需要通信的模块 a 和 b 中注入这个对象即可。

基于事件

这个又分两种方式

第一种是借助父controller。在子controller 中向父controller 触发（$emit）一个事件，然后在父 controller 中监听（$on）事件，再广播（$broadcast）给子 controller ，这样通过事件携带的参数，实现了数据经过父controller，在同级controller 之间传播。

第二种是借助$rootScope。每个Angular 应用默认有一个根作用域$rootScope， 根作用域位于最顶层，从它往下挂着各级作用域。所以，如果子控制器直接使用$rootScope 广播和接收事件，那么就可实现同级之间的通信。

详见AngularJS 中 Controller 之间的通信

18，如何看待**angular2**

相比Angular1.x，Angular2的改动很大，几乎算是一个全新的框架。

基于TypeScript（可以使用TypeScript 进行开发），在大型项目团队协作时，强语言类型更有利。

组件化，提升开发和维护的效率。

还有module 支持动态加载，new router，promise的原生支持等等。

迎合未来标准，吸纳其他框架的优点，值得期待，不过同时要学习的东西也更多了（ES next、TS、Rx等）。

Vue篇

**1**，**vue**的生命周期？

vue的生命周期主要分为几个简单，数据初始化，dom挂载，数据更新，组件卸载，在一个就是开启了组件缓存的时候，会有组件启用和组件停用阶段，每个阶段都去前后两个钩子除了缓存的那俩

数据初始化阶段

beforeCreate：在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

created：实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测 (data observer)，属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

dom挂载阶段

beforeMount：在挂载开始之前被调用：相关的 render 函数首次被调用。

mounted：el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

mounted 不会承诺所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以用 vm.$nextTick 替换掉 mounted

数据跟新阶段

beforeUpdate：数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。

updated：由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态。如果要相应状态改变，通常最好使用计算属性或 watcher 取而代之。

updated 不会承诺所有的子组件也都一起被重绘。如果你希望等到整个视图都重绘完毕，可以用 vm.$nextTick 替换掉 updated：

缓存启用的时候会有下面两个钩子

activated：keep-alive 组件激活时调用。

deactivated：keep-alive 组件停用时调用。

组件卸载的时候：

beforeDestroy：实例销毁之前调用。在这一步，实例仍然完全可用。

destroyed：Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

**2**，**vue**实现数据双向绑定的原理

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，触发相应的监听回调。具体话其实就是通过Obeject.defineProperty() 中的setter和getter来监听属性变动实现Observer进行数据的监听然后就是通知订阅者，那么订阅者其实就是简单的一个数组，这个数组中的内容就是我门使用了的一个数据的集合，使用了的数据可以通过getter得到，其实就是在调用的时候给数组里面添加一个订阅者这样就是实现了一个Watcher（需要监听的数据的集合），然后在实现一个Compile其作用就是解析模板指令，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，其实vue的数据双向绑定就是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。

3，vue路由的实现原理

在vue中路由主要有 hash与History interface两种方式实现前端路由，单页路由的特点就是采用前端路由系统，通过改变URL，在不重新请求页面的情况下，更新页面视图。目前在浏览器环境中这一功能的实现主要有两种方式：hash和History interface，这两种模式的实现分别是

先说hash，在浏览器的url中hash（“#”）符号的本来作用是加在URL中指示网页中的位置：#符号本身以及它后面的字符称之为hash，可通过window.location.hash属性读取。它具几个特点就是：hash虽然出现在URL中，但不会被包括在HTTP请求中。它是用来指导浏览器动作的，对服务器端完全无用，因此，改变hash不会重新加载页面，并且可以为hash的改变添加hashchange监听事件，在一个就是每一次改变hash（window.location.hash），都会在浏览器的访问历史中增加一个记录，我就可以通过他的这几个特点实现一个hash模式的单页路由，通过对location.hash的修改实现push方法（跳转页面），通过对location.href的修改实现replace()方法，通过对hashchange事件的监听实现页面跳转后的数据更新

而History模式则是完全采用了h5的新特性，从HTML5开始，History interface提供了两个新的方法：pushState(), replaceState()使得我们可以对浏览器历史记录栈进行修改，以及popState事件可以监听到状态的变更

不过history模式有一个问题就是

对于单页应用来讲，理想的使用场景是仅在进入应用时加载index.html，后续在的网络操作通过Ajax完成，不会根据URL重新请求页面，但是如果用户直接在地址栏中输入并回车，浏览器重启重新加载的时候history模式则会将URL修改得就和正常请求后端的URL一样，在此情况下重新向后端发送请求，如后端没有配置对应 的路由处理，则会返回404错误。这种问题的解决，一般情况下我们都是在后端进行配置，将所有的路由请求都指向index.html文件

4，vue组件的通讯？

1.父组件传子组件

  单项数据流props

  使用refs访问子组件

  使用$parent访问父组件

  使用$children访问子组件

  使用$root访问当前组件树的根 Vue 实例

  在template元素使用scope 可以访问到子组件中slot元素的属性值  要13k

2.子组件传递数据给父组件：

  子组件通过事件给父组件传数据，子组件通过$emit（eventName）触发事件，父组件通过$on监听事件

3.兄弟组件间通信

①用事件发布订阅 var bus=new vue（）；bus.$emit("id-selected",1)；bus.$on（"id-selected",function(id){}）

②用vuex来实现

**5**，**VueX**是做什么的？

vue components(组件) -> action（只能通过 dispatch 调用，与此同时可以异步与后端的 API 做交互） -> mutations(只能通过 action 发起 commit 调用，此时开发工具可以监测到数据的流动) -> state （mutations 传递修改过的状态到 state）-> state 自动同步到视图

Vuex 是适用于 Vue.js 应用的状态管理库，为应用中的所有组件提供集中式的状态存储与操作，保证了所有状态以可预测的方式进行修改；

        state: state 定义了应用状态的数据结构，同样可以在这里设置默认的初始状态。

        actions:Actions 即是定义提交触发更改信息的描述，常见的例子有从服务端获取数据，在数据获取完成后会调用store.commit()来调用更改 Store 中的状态。可以在组件中使用dispatch来发出 Actions。

        mutations: 调用 mutations 是唯一允许更新应用状态的地方。

        getters: Getters 允许组件从 Store 中获取数据，譬如我们可以从 Store 中的 projectList 中筛选出已完成的项目列表

        modules: modules 对象允许将单一的 Store 拆分为多个 Store 的同时保存在单一的状态树中。随着应用复杂度的增加，这种拆分能够更好地组织代码

但是vuex也有缺点就是，vuex中保存的数据是和网页的生命周期同步的，当执行页面刷新的时候vuex中所有数据都会消失复位到初始状态，所以不太适合做有分享页面的数据交互（在这种项目中vuex只适合数据的集中管理，不适合数据的存储，这种情况一般是使用路由传递参数会好一些），适合后台管理系统多一些，后台管理系统一般都是公司内部使用；

**6**，**Vue**的路由如何传参？

<router-link :to="{path:‘details‘,query: {id:el.tog\_line\_id}}">

<router-link :to="{name:‘details‘,params: {id:el.tog\_line\_id}}”>

然后在组件内部通过$route.params来读取数据，但是路由传递的参数值是对象的话就不行了会报错，一般都是在传递之前先吧需要传递的数据使用base64转化一下就不会造成路由报错了

**7**，**Vue**和**angular**的区别？

    1.在 API 与设计两方面上 Vue.js 都比 Angular 简单得多，因此你可以快速地掌握它的全部特性并投入开发。

    2.Vue.js 是一个更加灵活开放的解决方案。它允许你以希望的方式组织应用程序，而不是任何时候都必须遵循 Angular 制定的规则。它仅仅是一个视图层，所以你可以将它嵌入一个现有页面而不一定要做成一个庞大的单页应用。在配合其他库方面它给了你更大的的空间，但相应，你也需要做更多的架构决策。例如，Vue.js 核心默认不包含路由和 Ajax 功能，并且通常假定你在应用中使用了一个模块构建系统。这可能是最重要的区别。

    3.Angular 使用双向绑定，Vue 也支持双向绑定，不过默认为单向绑定，数据从父组件单向传给子组件。在大型应用中使用单向绑定让数据流易于理解。

    4.在 Vue.js 中指令和组件分得更清晰。指令只封装 DOM 操作，而组件代表一个自给自足的独立单元 —— 有自己的视图和数据逻辑。在 Angular 中两者有不少相混的地方。

    5.Vue.js 有更好的性能，并且非常非常容易优化，因为它不使用脏检查。Angular，当 watcher 越来越多时会变得越来越慢，因为作用域内的每一次变化，所有 watcher 都要重新计算。并且，如果一些 watcher 触发另一个更新，脏检查循环（digest cycle）可能要运行多次。 Angular 用户常常要使用深奥的技术，以解决脏检查循环的问题。有时没有简单的办法来优化有大量 watcher 的作用域。Vue.js 则根本没有这个问题，因为它使用基于依赖追踪的观察系统并且异步列队更新，所有的数据变化都是独立地触发，除非它们之间有明确的依赖关系。唯一需要做的优化是在 v-for 上使用 track-by。

**8**，**Vue**和**SEO**？

vue主要一个开发框架，通过vue－router可以实现单页应用的开发，但是单页应用本身就对seo不友好，现在vue官方给出的解决方案是vue-server-renderer服务端渲染，但是我觉得对于一个真正适合做成单页应用的项目的话，seo其实是可以不用考虑的，反正项目本身是需要登录才能进入的，但是像其他一些类似于商城的项目的话，我们可以吧项目分开，像单个的详情页我们没有必要放到单页路由中，可以把她们分出去，毕竟对于商城来说商品才是详情才是需要seo优化的。或者我们完全没有必要用了vue就得吧项目做成单页的，我们可以使用一些他的特性，毕竟我觉得vue给我带来最大的好处就是组件化开发；

**9**，**Vue**的优缺点？

优点：

简单：官方文档很清晰，比 Angular 简单易学。

快速：异步批处理方式更新 DOM。组合：用解耦的、可复用的组件组合你的应用程序。

紧凑：~18kb min+gzip，且无依赖。

强大：表达式 & 无需声明依赖的可推导属性 (computed properties)。

对模块友好：可以通过 NPM、Bower 或 Duo 安装，不强迫你所有的代码都遵循 Angular 的各种规定，使用场景更加灵活。

vue 作者是中国人。哈哈

缺点：

不兼容ie8，不过总体上觉得vue对我来说还是非常不错的，能看的懂文档，能上的去手，开发模式也比较好

**10**，**Vue**的脚手架有几种？

5种 simple,webpack-simple,vue-cli,browserify,browser-simple

**11**，指令**keep-alive**

指令keep-alive是做组件缓存的，就是把组件缓存起来不会销毁，下次打开这个组件直接显示出来，如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。一般用于有上啦加载的页面，因为如果一个页面有上啦加载的话，用户加载了好几页了都然后点击进入详情，在返回又得重新加载，这个体验很不好，有了keep-alive后就可以吧这个列表页面保存在内存中避免用户的重复操作，增强用户体验

**12.**如何让**css**只在当前组件中起作用

在每一个vue组件中都可以定义各自的css，js，如果希望组件内写的css只对当前组件起作用，只需要在style中写入scoped，即：

<style scoped></style>

**13.Vuejs**在变化检测问题

1.检测数组

由于javascript的限制，vuejs不能检测到下面数组的变化：

直接索引设置元素，如vm.item[0]={};

修改数据的长度，如vm.item.length。

为了解决问题1，Vuejs扩展了观察数组，为它添加一个$set()方法：

// 与 `example1.items[0] = ...` 相同，但是能触发视图更新

example1.items.$set(0, { childMsg: 'Changed!'})

问题2，需要一个空数组替换items。

除了$set()，vuejs也为观察数组添加了$remove()方法，用于从目标数组中查找并删除元素，在内部调用了splice()。因此，不必：

var index = this.items.indexOf(item)

if (index !== -1) {

  this.items.splice(index, 1)

}

只需：

this.items.$remove(item);

2.检测对象

受ES5的显示，Vuejs不能检测到对象属性的添加或删除。因为Vuejs在初始化时候将属性转化为getter/setter，所以属性必须在data对象才能让Vuejs转换它，才能让它是响应的，例如：

var data = { a: 1 }

var vm = new Vue({

  data: data

})

// `vm.a` 和 `data.a` 现在是响应的

vm.b = 2

// `vm.b` 不是响应的

data.b = 2

// `data.b` 不是响应的

不过，有办法在实例创建之后添加属性并且让它是响应的。对于Vue实例，可以使用$set(key,value)实例方法：

vm.$set('b', 2)

// `vm.b` 和 `data.b` 现在是响应的

对于普通数据对象，可以使用全局方法Vue.set(object, key, value):

Vue.set(data, 'c', 3)

// `vm.c` 和 `data.c` 现在是响应的

有时你想向已有对象上添加一些属性，例如使用 Object.assign() 或 \_.extend() 添加属性。但是，添加到对象上的新属性不会触发更新。这时可以创建一个新的对象，包含原对象的属性和新的属性：

// 不使用 `Object.assign(this.someObject, { a: 1, b: 2 })`

this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

**14.**关于**vuejs**页面闪烁**{{message}**}

在vuejs指令中有v-cloak，这个指令保持在元素上直到关联实例结束编译。和CSS规则如[v-cloak]{display:none}一起用时，这个指令可以隐藏未编译的Mustache标签直到实例准备完毕。用法如下：

[v-cloak]{

    display:none;

}

<div v-cloak>{{message}}</div>

这样<div>不会显示，直到编译结束

**15.**组件命名的约定

当注册组件（或者props）时，可以使用 kebab-case ，camelCase ，或 TitleCase

// 在组件定义中

components: {

  // 使用 kebab-case 形式注册

  'kebab-cased-component': { /\* ... \*/ },

  // register using camelCase

  'camelCasedComponent': { /\* ... \*/ },

  // register using TitleCase

  'TitleCasedComponent': { /\* ... \*/ }

}

在 HTML 模版中，只能使用 kebab-case 形式：

<!-- 在HTML模版中始终使用 kebab-case -->

<kebab-cased-component></kebab-cased-component>

<camel-cased-component></camel-cased-component>

<title-cased-component></title-cased-component>

当使用字符串模式时可以使用 camelCase 、 TitleCase 或者 kebab-case 来引用：

<!-- 在字符串模版中可以用任何你喜欢的方式! -->

<my-component></my-component>

<myComponent></myComponent>

<MyComponent></MyComponent>

**16.key**的使用

一般情况下，vue在渲染完成后，如果数据发生变化，只会重新渲染数据，不会重新渲染整个元素，但是有时候我们需要元素被重新渲染，此时就需要使用key关键字，使用v-bind绑定key关键字，可以实现在数据发生变化时候重新渲染整个元素。注：同一父级元素下所有子元素如果都要在数据变化后重新渲染元素，则需要被绑定的key

**17**，**$route**和**$router**的区别

$route是路由信息对象，包含了path，params，query，hash，fullPath，matched，name这些路由信息参数

而$router是路由实例对象包含了路由的一些跳转方法，钩子函数等

**18**，**vue**路由的钩子函数

官方称为导航守卫可以控制导航的跳转，有beforeEach，afterEach等，一般用于页面title的修改，一些需要登录才能调整页面的重定向等功能，beforeEach主要有3个参数to, from, next

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。可以控制网页的跳转

**19**，**watch**，**computed**，**methods**的区别？

watch是做单一的数据监听，方法名字必须和需要监听的数据同名，不需要返回值，可以得到数据修改前后修改后的值

computed和methods基本相同，都是方法，都是在数据改变的时候如果方法内部有该数据的依赖都是自动执行，

但是它们俩定位不同，computed是计算属性，而methods是放的操作方法，

还有就是调用形式不同，计算属性调用不需要加括弧，methods则需要加（）

在一个就是computed会将计算的值进行缓存，如果方法内部依赖的值没有发生变化调用computed不管多少次都只是执行一次，其他调用反回的是缓存的值，而methods则调用多少次就会执行多少次

**20**，**vue**中**sync**的用处？

vue中sync其实就是自定义事件的一个语法糖

<comp :foo.sync="bar"></comp>

会被扩展为：

<comp :foo="bar" @update:foo="val => bar = val"></comp>

当子组件需要更新 foo 的值时，它需要显式地触发一个更新事件：

this.$emit('update:foo', newValue)

所以他的好处就是简化了自定义事件的写法。

**21**，**vue**组件怎么划分？

按着页面的框架去划分，将页面中的每一块都可以划分成为一个组件，然后从中提取公共组件，一般都是将组件分为页面组件和公共组件

22，vue的插件

23，vue的自定义指令

React面试题

**1**，调用 **setState** 之后发生了什么？

在代码中调用setState函数之后，React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程（Reconciliation）。

经过调和过程，React 会以相对高效的方式根据新的状态构建 React 元素树并且着手重新渲染整个UI界面。

在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染。

在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

**2**，**React** 中 **Element** 与 **Component** 的区别是？

简单而言，React Element 是描述屏幕上所见内容的数据结构，是对于 UI 的对象表述。典型的 React Element 就是利用 JSX 构建的声明式代码片然后被转化为createElement的调用组合。而 React Component 则是可以接收参数输入并且返回某个 React Element 的函数或者类。更多介绍可以参考React Elements vs React Components。

**3**，在什么情况下你会优先选择使用 **Class Component** 而不是 **Functional Component**？

在组件需要包含内部状态或者使用到生命周期函数的时候使用 Class Component ，否则使用函数式组件。

**4**，**React** 中 **refs** 的作用是什么？

this.refs.textInputRefs 是 React 提供给我们的安全访问 DOM 元素或者某个组件实例的句柄。

我们可以为元素添加ref属性然后在回调函数中接受该元素在 DOM 树中的句柄，该值会作为回调函数的第一个参数返回：

class CustomForm extends Component {

    handleSubmit = () => {

        console.log("Input Value: ", this.input.value)

    }

    render () {

        return (

            <form onSubmit={this.handleSubmit}>

                <input  type='text' ref={(input) => this.input = input} />

                <button type='submit'>Submit</button>

            </form>

        )

    }

}

上述代码中的input域包含了一个ref属性，该属性声明的回调函数会接收input对应的 DOM 元素，我们将其绑定到this指针以便在其他的类函数中使用。另外值得一提的是，refs 并不是类组件的专属，函数式组件同样能够利用闭包暂存其值：

function CustomForm ({handleSubmit}) {

    let inputElement

    return (

        <form onSubmit={() => handleSubmit(inputElement.value)}>

            <input type='text' ref={(input) => inputElement = input} />

            <button type='submit'>Submit</button></form>

       )

}

**5**，**React** 中 **keys** 的作用是什么？

Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。

render () {

    return (

        <ul>{this.state.todoItems.map(({task, uid}) => {

                return <li key={uid}>{task}</li>

          })}</ul>

)}

在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。在 React Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染。此外，React 还需要借助 Key 值来判断元素与本地状态的关联关系，因此我们绝不可忽视转换函数中 Key 的重要性。

Controlled Component 与 Uncontrolled Component 之间的区别是什么？

React 的核心组成之一就是能够维持内部状态的自治组件，不过当我们引入原生的HTML表单元素时（input,select,textarea 等），我们是否应该将所有的数据托管到 React 组件中还是将其仍然保留在 DOM 元素中呢？

这个问题的答案就是受控组件与非受控组件的定义分割。受控组件（Controlled Component）代指那些交由 React 控制并且所有的表单数据统一存放的组件。

譬如下面这段代码中username变量值并没有存放到DOM元素中，而是存放在组件状态数据中。

任何时候我们需要改变username变量值时，我们应当调用setState函数进行修改。class ControlledForm extends Component {  state = {    username: ''  }  updateUsername = (e) => {    this.setState({      username: e.target.value,    })  }  handleSubmit = () => {}  render () {    return (<form onSubmit={this.handleSubmit}><input          type='text'          value={this.state.username}          onChange={this.updateUsername} /><button type='submit'>Submit</button></form>    )  }}而非受控组件（Uncontrolled Component）则是由DOM存放表单数据，并非存放在 React 组件中。我们可以使用 refs 来操控DOM元素：class UnControlledForm extends Component {  handleSubmit = () => {    console.log("Input Value: ", this.input.value)  }  render () {    return (<form onSubmit={this.handleSubmit}><input          type='text'          ref={(input) => this.input = input} /><button type='submit'>Submit</button></form>    )  }}竟然非受控组件看上去更好实现，我们可以直接从 DOM 中抓取数据，而不需要添加额外的代码。不过实际开发中我们并不提倡使用非受控组件，因为实际情况下我们需要更多的考虑表单验证、选择性的开启或者关闭按钮点击、强制输入格式等功能支持，而此时我们将数据托管到 React 中有助于我们更好地以声明式的方式完成这些功能。引入 React 或者其他 MVVM 框架最初的原因就是为了将我们从繁重的直接操作 DOM 中解放出来。

React 下一代调和算法 Fiber 会通过开始或停止渲染的方式优化应用性能，其会影响到 componentWillMount 的触发次数。对于 componentWillMount 这个生命周期函数的调用次数会变得不确定，React 可能会多次频繁调用 componentWillMount。如果我们将 AJAX 请求放到 componentWillMount 函数中，那么显而易见其会被触发多次，自然也就不是好的选择。

**7**，在生命周期中的哪一步你应该发起 **AJAX** 请求？

我们应当将AJAX 请求放到 componentDidMount 函数中执行，主要原因有下：

如果我们将 AJAX 请求放置在生命周期的其他函数中，我们并不能保证请求仅在组件挂载完毕后才会要求响应。如果我们的数据请求在组件挂载之前就完成，并且调用了setState函数将数据添加到组件状态中，对于未挂载的组件则会报错。而在 componentDidMount 函数中进行 AJAX 请求则能有效避免这个问题。

**8**，**shouldComponentUpdate** 的作用是啥以及为何它这么重要？

shouldComponentUpdate 允许我们手动地判断是否要进行组件更新，根据组件的应用场景设置函数的合理返回值能够帮我们避免不必要的更新。

**9**，如何告诉 **React** 它应该编译生产环境版本？

通常情况下我们会使用 Webpack 的 DefinePlugin 方法来将 NODE\_ENV 变量值设置为 production。编译版本中 React 会忽略 propType 验证以及其他的告警信息，同时还会降低代码库的大小，React 使用了 Uglify 插件来移除生产环境下不必要的注释等信息。

**10**，概述下 **React** 中的事件处理逻辑

为了解决跨浏览器兼容性问题，React 会将浏览器原生事件（Browser Native Event）封装为合成事件（SyntheticEvent）传入设置的事件处理器中。这里的合成事件提供了与原生事件相同的接口，不过它们屏蔽了底层浏览器的细节差异，保证了行为的一致性。另外有意思的是，React 并没有直接将事件附着到子元素上，而是以单一事件监听器的方式将所有的事件发送到顶层进行处理。这样 React 在更新 DOM 的时候就不需要考虑如何去处理附着在 DOM 上的事件监听器，最终达到优化性能的目的。

**11**，**createElement** 与 **cloneElement** 的区别是什么？

createElement 函数是 JSX 编译之后使用的创建 React Element 的函数，而 cloneElement 则是用于复制某个元素并传入新的 Props。

**12**，传入 **setState** 函数的第二个参数的作用是什么？

是个函数，该函数会在setState函数调用完成并且组件开始重渲染的时候被调用，我们可以用该函数来监听渲染是否完成：

this.setState(

  { username: 'tylermcginnis33' },

  () => console.log('setState has finished and the component has re-rendered.')

)

下述代码有错吗？

this.setState((prevState, props) => {  return {    streak: prevState.streak + props.count  }})这段代码没啥问题，不过只是不太常用罢了

23.react中state与props的区别?

ajax面试题

1，Ajax 是什么? 如何创建一个Ajax？

ajax的全称：Asynchronous Javascript And XML。异步传输+js+xml。

所谓异步，在这里简单地解释就是：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

            (1)创建XMLHttpRequest对象,也就是创建一个异步调用对象

            (2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息

            (3)设置响应HTTP请求状态变化的函数

            (4)发送HTTP请求

            (5)获取异步调用返回的数据

            (6)使用JavaScript和DOM实现局部刷新

            ajax是一种创建交互式网页的计算

2，同步和异步的区别?

同步的概念应该是来自于OS中关于同步的概念:不同进程为协同完成某项工作而在先后次序上调整(通过阻塞,唤醒等方式).同步强调的是顺序性.谁先谁后.异步则不存在这种顺序性.

同步：浏览器访问服务器请求，用户看得到页面刷新，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,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。

3，如何解决跨域问题?

jsonp、 iframe、window.name、window.postMessage、服务器上设置代理页面

参考：http://qiutc.me/post/cross-domain-collections.html

4，http状态码有那些？分别代表是什么意思？

简单版

             [

                        100  Continue   继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

                        200  OK         正常返回信息

                        201  Created    请求成功并且服务器创建了新的资源

                        202  Accepted   服务器已接受请求，但尚未处理

                        301  Moved Permanently  请求的网页已永久移动到新位置。

                        302 Found       临时性重定向。

                        303 See Other   临时性重定向，且总是使用 GET 请求新的 URI。

                        304  Not Modified 自从上次请求后，请求的网页未修改过。

                        400 Bad Request  服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

                        401 Unauthorized 请求未授权。

                        403 Forbidden   禁止访问。

                        404 Not Found   找不到如何与 URI 相匹配的资源。

                        500 Internal Server Error  最常见的服务器端错误。

                        503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

             ]

5，一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？（流程说的越详细越好）

注：这个问题最主要的是问浏览器的渲染原理

而高手可以根据自己擅长的领域自由发挥，从URL规范、HTTP协议、DNS、CDN、数据库查询、到浏览器流式解析、CSS规则构建、layout、paint、onload/domready、JS执行、JS API绑定等等；

详细版：

            1、浏览器会开启一个线程来处理这个请求，对 URL 分析判断如果是 http 协议就按照 Web 方式来处理;

            2、调用浏览器内核中的对应方法，比如 WebView 中的 loadUrl 方法;

            3、通过DNS解析获取网址的IP地址，设置 UA 等信息发出第二个GET请求;

            4、进行HTTP协议会话，客户端发送报头(请求报头);

            5、进入到web服务器上的 Web Server，如 Apache、Tomcat、Node.JS 等服务器;

            6、进入部署好的后端应用，如 PHP、Java、JavaScript、Python 等，找到对应的请求处理;

            7、处理结束回馈报头，此处如果浏览器访问过，缓存上有对应资源，会与服务器最后修改时间对比，一致则返回304;

            8、浏览器开始下载html文档(响应报头，状态码200)，同时使用缓存;

             9、文档树建立，根据标记请求所需指定MIME类型的文件（比如css、js）,同时设置了cookie;

            10、页面开始渲染DOM，JS根据DOM API操作DOM,执行事件绑定等，页面显示完成。

简洁版：

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

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

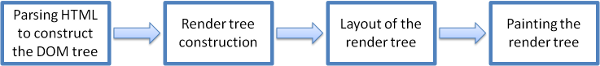
            浏览器对加载到的资源（HTML、JS、CSS等）进行语法解析，建立相应的内部数据结构（如HTML的DOM）；

            载入解析到的资源文件，渲染页面，完成。

浏览器的渲染原理

渲染引擎首先通过网络获得所请求文档的内容，通常以8K分块的方式完成。下面是渲染引擎在取得内容之后的基本流程：

　　解析html以构建dom树 -> 构建render树 -> 布局render树 -> 绘制render树



这里先解释一下几个概念，方便大家理解：

　　DOM Tree：浏览器将HTML解析成树形的数据结构。

　　CSS Rule Tree：浏览器将CSS解析成树形的数据结构。

　　Render Tree: DOM和CSSOM合并后生成Render Tree。

　　layout: 有了Render Tree，浏览器已经能知道网页中有哪些节点、各个节点的CSS定义以及他们的从属关系，从而去计算出每个节点在屏幕中的位置。

　　painting: 按照算出来的规则，通过显卡，把内容画到屏幕上。

　　reflow（回流）：当浏览器发现某个部分发生了点变化影响了布局，需要倒回去重新渲染，内行称这个回退的过程叫 reflow。reflow 会从 <html> 这个 root frame 开始递归往下，依次计算所有的结点几何尺寸和位置。reflow 几乎是无法避免的。现在界面上流行的一些效果，比如树状目录的折叠、展开（实质上是元素的显 示与隐藏）等，都将引起浏览器的 reflow。鼠标滑过、点击……只要这些行为引起了页面上某些元素的占位面积、定位方式、边距等属性的变化，都会引起它内部、周围甚至整个页面的重新渲 染。通常我们都无法预估浏览器到底会 reflow 哪一部分的代码，它们都彼此相互影响着。

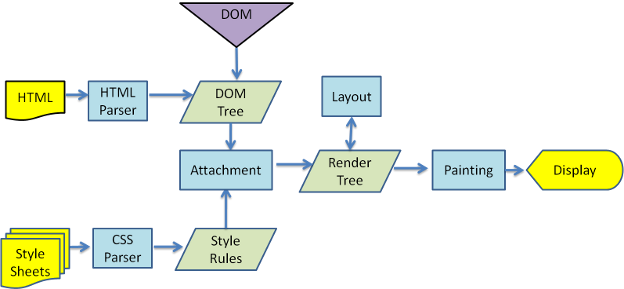
　　repaint（重绘）：改变某个元素的背景色、文字颜色、边框颜色等等不影响它周围或内部布局的属性时，屏幕的一部分要重画，但是元素的几何尺寸没有变。

注意：*(1)display:none* 的节点不会被加入*Render Tree*，而*visibility: hidden* 则会，所以，如果某个节点最开始是不显示的，设为*display:none*是更优的。

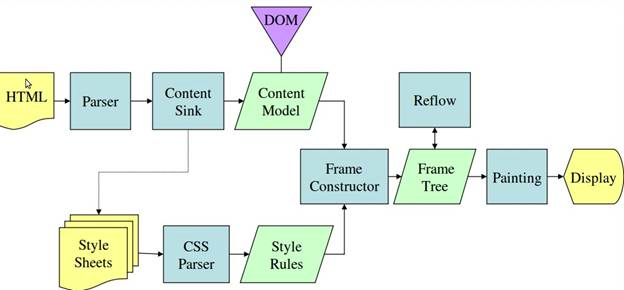
*(2)display:none* 会触发 *reflow*，而 *visibility:hidden* 只会触发 *repaint*，因为没有发现位置变化。

*(3)*有些情况下，比如修改了元素的样式，浏览器并不会立刻*reflow* 或 *repaint* 一次，而是会把这样的操作积攒一批，然后做一次 *reflow*，这又叫异步 *reflow* 或增量异步 *reflow*。但是在有些情况下，比如*resize* 窗口，改变了页面默认的字体等。对于这些操作，浏览器会马上进行 *reflow*。

　　来看看webkit的主要流程：



　　再来看看Geoko的主要流程：



　　Gecko 里把格式化好的可视元素称做“帧树”（Frame tree）。每个元素就是一个帧（frame）。 webkit 则使用”渲染树”这个术语，渲染树由”渲染对象”组成。webkit 里使用”layout”表示元素的布局，Gecko则称为”reflow”。Webkit使用”Attachment”来连接DOM节点与可视化信息以构建渲染树。一个非语义上的小差别是Gecko在HTML与DOM树之间有一个附加的层 ，称作”content sink”，是创建DOM对象的工厂。

　　尽管Webkit与Gecko使用略微不同的术语，这个过程还是基本相同的，如下：

　　1. 浏览器会将HTML解析成一个DOM树，DOM 树的构建过程是一个深度遍历过程：当前节点的所有子节点都构建好后才会去构建当前节点的下一个兄弟节点。

　　2. 将CSS解析成 CSS Rule Tree 。

　　3. 根据DOM树和CSSOM来构造 Rendering Tree。注意：Rendering Tree 渲染树并不等同于 DOM 树，因为一些像Header或display:none的东西就没必要放在渲染树中了。

　　4. 有了Render Tree，浏览器已经能知道网页中有哪些节点、各个节点的CSS定义以及他们的从属关系。下一步操作称之为layout，顾名思义就是计算出每个节点在屏幕中的位置。

　　5. 再下一步就是绘制，即遍历render树，并使用UI后端层绘制每个节点。

　　注意：上述这个过程是逐步完成的，为了更好的用户体验，渲染引擎将会尽可能早的将内容呈现到屏幕上，并不会等到所有的*html*都解析完成之后再去构建和布局*render*树。它是解析完一部分内容就显示一部分内容，同时，可能还在通过网络下载其余内容。

8，请解释 JSONP 的工作原理，以及它为什么不是真正的 AJAX。

jsonp的工作原理是，动态的创建了一个全局方法，并且动态生成script标签，将script标签的src属性变为（接口地址?callback=动态生成方法的方法名）请求数据，而后台则需要将接收到的callback值与数据一同返回，呈现出执行js方法的语句（方法名(数据)），其实就是在请求回来的数据中是执行请求是动态生成的js方法，生成了假象的ajax，所以jsonp只能做get类型请求

9，请解释一下 JavaScript 的同源策略。

同源策略是客户端脚本（尤其是Javascript）的重要的安全度量标准。它最早出自Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。

它的精髓很简单：它认为自任何站点装载的信赖内容是不安全的。当被浏览器半信半疑的脚本运行在沙箱时，它们应该只被允许访问来自同一站点的资源，而不是那些来自其它站点可能怀有恶意的资源。

这里的同源指的是：同协议，同域名和同端口

10，为什么要有同源限制？

我们举例说明：比如一个黑客程序，他利用IFrame把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。

11，创建ajax过程

(1)创建`XMLHttpRequest`对象,也就是创建一个异步调用对象.

(2)创建一个新的`HTTP`请求,并指定该`HTTP`请求的方法、`URL`及验证信息.

(3)设置响应`HTTP`请求状态变化的函数.

(4)发送`HTTP`请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

            var xmlHttp = new XMLHttpRequest();

            xmlHttp.open('GET','demo.php','true');

            xmlHttp.send()

            xmlHttp.onreadystatechange = function(){

                        if(xmlHttp.readyState === 4 & xmlHttp.status === 200){

                        }

            }

11，常见web安全及防护原理

sql注入原理

就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

总的来说有以下几点：

1.永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.永远不要使用动态拼装SQL，可以使用参数化的SQL或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

XSS原理及防范

Xss(cross-site scripting)攻击指的是攻击者往Web页面里插入恶意 html标签或者javascript代码。比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取cookie中的用户私密信息；或者攻击者在论坛中加一个恶意表单当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点。

XSS防范方法

首先代码里对用户输入的地方和变量都需要仔细检查长度和对”<”,”>”,”;”,”’”等字符做过滤；其次任何内容写到页面之前都必须加以encode，避免不小心把html tag 弄出来。这一个层面做好，至少可以堵住超过一半的XSS 攻击。首先，避免直接在cookie 中泄露用户隐私，例如email、密码等等。其次，通过使cookie 和系统ip 绑定来降低cookie 泄露后的危险。这样攻击者得到的cookie 没有实际价值，不可能拿来重放。如果网站不需要再浏览器端对cookie 进行操作，可以在Set-Cookie 末尾加上HttpOnly 来防止javascript 代码直接获取cookie 。尽量采用POST 而非GET 提交表单

12，XSS与CSRF有什么区别吗？

XSS是获取信息，不需要提前知道其他用户页面的代码和数据包。CSRF是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。

要完成一次CSRF攻击，受害者必须依次完成两个步骤：

登录受信任网站A，并在本地生成Cookie。

在不登出A的情况下，访问危险网站B。

CSRF的防御

服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。

通过验证码的方法

13，HTTP和HTTPS

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

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

14，为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性

15，GET和POST的区别，何时使用POST？

GET：一般用于信息获取，使用URL传递参数，对所发送信息的数量也有限制，一般在2000个字符

POST：一般用于修改服务器上的资源，对所发送的信息没有限制。

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值，也就是说Get是通过地址栏来传值，而Post是通过提交表单来传值。

然而，在以下情况中，请使用 POST 请求：

无法使用缓存文件（更新服务器上的文件或数据库）

向服务器发送大量数据（POST 没有数据量限制）

发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

16，ajax的缺点和在IE下的问题？

ajax的缺点

    1、ajax不支持浏览器back按钮。

    2、安全问题 AJAX暴露了与服务器交互的细节。

    3、对搜索引擎的支持比较弱。

    4、破坏了程序的异常机制。

    5、不容易调试。

IE缓存问题

在IE浏览器下，如果请求的方法是GET，并且请求的URL不变，那么这个请求的结果就会被缓存。解决这个问题的办法可以通过实时改变请求的URL，只要URL改变，就不会被缓存，可以通过在URL末尾添加上随机的时间戳参数('t'= + new Date().getTime())

或者：

    open('GET','demo.php?rand=+Math.random()',true);//

    Ajax请求的页面历史记录状态问题

    可以通过锚点来记录状态，location.hash。让浏览器记录Ajax请求时页面状态的变化。

    还可以通过HTML5的history.pushState，来实现浏览器地址栏的无刷新改变

19，前端需要注意哪些SEO

合理的title、description、keywords：搜索对着三项的权重逐个减小，title值强调重点即可，重要关键词出现不要超过2次，而且要靠前，不同页面title要有所不同；description把页面内容高度概括，长度合适，不可   过分堆砌关键词，不同页面description有所不同；keywords列举出重要关键词即可

语义化的HTML代码，符合W3C规范：语义化代码让搜索引擎容易理解网页

重要内容HTML代码放在最前：搜索引擎抓取HTML顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容一定会被抓取

重要内容不要用js输出：爬虫不会执行js获取内容

少用iframe：搜索引擎不会抓取iframe中的内容

非装饰性图片必须加alt

提高网站速度：网站速度是搜索引擎排序的一个重要指标

20.什么是Ajax和JSON，它们的优缺点。

Ajax是异步JavaScript和XML，用于在Web页面中实现异步数据交互。

优点：

    可以使得页面不重载全部内容的情况下加载局部内容，降低数据传输量

    避免用户不断刷新或者跳转页面，提高用户体验

缺点：

    对搜索引擎不友好（

    要实现ajax下的前后退功能成本较大

    可能造成请求数的增加

    跨域问题限制

JSON是一种轻量级的数据交换格式，ECMA的一个子集

优点：轻量级、易于人的阅读和编写，便于机器（JavaScript）解析，支持复合数据类型（数组、对象、字符串、数字）

21，AJAX的请求方式？

GET：GET是http的默认请求方式， 一般用来获取数据。

HEAD：HEAD方法与GET方法一样，都是向服务器发出指定资源的请求。但是，服务器在响应HEAD请求时不会回传资源的内容部分，即：响应主体。这样，我们可以不传输全部内容的情况下，就可以获取服务器的响应头信息。HEAD方法常被用于客户端查看服务器的性能。

POST：POST请求会向指定资源提交数据，请求服务器进行处理。如：表单提交、文件上传。

PUT：PUT请求会身向指定资源位置上传其最新内容，通过该方法客户端可以将指定资源的最新数据传送给服务器取代指定的资源的内容。

DELETE：DELETE请求用于请求服务器删除所请求URI所标识的资源。DELETE请求后指定资源会被删除。

TRACE：返回接受到的请求，用来查看数据经过中间服务器时发生了哪些变动。

OPTIONS：OPTIONS请求与HEAD类似，一般也是用于客户端查看服务器的性能。 这个方法会请求服务器返回该资源所支持的所有HTTP请求方法，该方法会用'\*'来代替资源名称，向服务器发送OPTIONS请求，可以测试服务器功能是否正常。JavaScript的XMLHttpRequest对象进行CORS跨域资源共享时，就是使用OPTIONS方法发送嗅探请求，以判断是否有对指定资源的访问权限。

CONNECT：要求使用SSL和TLS进行TCP通信。

PATCH：请求修改局部数据

**22**，**RESTful**架构

REST是一种架构风格：无状态，以资源为中心，充分利用HTTP协议和URI协议，提供统一的接口定义，使得它作为一种设计Web服务的方法而变得流行。在某种意义上，通过强调URI和HTTP等早期Internet标准，REST是对大型应用程序服务器时代之前的Web方式的回归。

架构约束：

客户-服务器:通信只能由客户端单方面发起，表现为请求-响应的形式。

无状态:通信的会话状态（Session State）应该全部由客户端负责维护。

缓存:响应内容可以在通信链的某处被缓存，以改善网络效率。

统一接口:通信链的组件之间通过统一的接口相互通信，以提高交互的可见性。

分层系统:通过限制组件的行为（即，每个组件只能"看到"与其交互的紧邻层），将架构分解为若干等级的层。

按需代码:支持通过下载并执行一些代码（例如Java Applet、Flash或JavaScript），对客户端的功能进行扩展。

主要特征：

面向资源（Resource Oriented）

可寻址（Addressability）

连通性（Connectedness）

无状态（Statelessness）

统一接口（Uniform Interface）

超文本驱动（Hypertext Driven）

**23**，**cookie** 和**session** 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

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

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

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

5、所以一般情况下将登陆信息等重要信息存放为SESSION其他信息如果需要保留，可以放在COOKIE中

**24**，**Cookie**和**localstorage**、**session storage**的区别

Cookie技术浏览器兼容性好，但操作比较复杂，需要程序员自己封装，源生的Cookie接口不友好,   存储的内容较小， cookie的数据会随着ajax的请求发送到服务端，一般情况主要用在用户登录的时候我们可以通过在 Cookie 中存入一段辨别用户身份的数据，用于后台判断。

WebStorage则不能超过8MB，操作简单；可以代替一些cookie的工作，一般主要是用于存储一些本地数据，购物车数据之类的在安全方面的话，都不安全，一般就是对数据进行一些简单的加密，如base64编码，加密约定之类的东西localstorage、sessionstorage一个是长期存储，一个是会话存储

**25**，**fetch**与**ajax**的区别？

Ajax主要是利用的是XMLHttpRequest对象来请求数据的。

Fetch是window的一个方法 主要特点是

1、第一个参数是URL

2、第二个参数可选参数 可以控制不同的init对象

3、使用了js 中的promise对象

fetch 的第二参数中

1、默认的请求为get请求 可以使用method:post 来进行配置

2、第一步中的 response有许多方法 json() text() formData()

3、Fetch跨域的时候默认不会带cookie ，如果你想在fetch请求里附带cookies之类的凭证信息，可以将 credentials参数设置成 “include” 值。还有就是fetch返回的promise在某些错误的http状态下如400、500等不会reject的错误状态，相反它会被resolve；只有网络错误会导致请求不能完成时，fetch 才会被 reject；所以一般会对fetch请求做一层封装，在resolve中真对于大于200和 小于300的状态返回正确信息，其他则返回错误信息所有的IE浏览器都不会支持 fetch()方法

26，浏览器本地存储

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和localStorage。

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

而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。

27，请你谈谈Cookie的弊端

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。

1，每个特定的域名下最多生成20个cookie

2，IE6或更低版本最多20个cookie

3，IE7和之后的版本最后可以有50个cookie。

4，Firefox最多50个cookie

5，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userdata，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

            1.通过良好的编程，控制保存在cookie中的session对象的大小。

            2.通过加密和安全传输技术（SSL），减少cookie被破解的可能性。

            3.只在cookie中存放不敏感数据，即使被盗也不会有重大损失。

            4.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：

            1.`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.

            2.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。

            3.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。

28，跨域有几种方法？

                  跨域的话我研究过的有两种方法，jsonp和CORS两种，

                  jsonp的话以前比较常用一些，主要实现原理就是通过引用js的方法来获取跨域资源，通过执行js函数的方法调用回调成功的函数，

                  cors的话这个比较简单一些，但是有一些兼容问题，只支持ie10+ 和现代浏览器适合用于移动端，实现原理就是在后端加上header头信息Access-Control-Allow-Origin: ＊，也可以指定域名访问Access-Control-Allow-Origin: http://api.qiutc.me

                  其他的方法暂时还没有研究过

29，jsonp的原理？

jsonp的实现原理就是，jquery中动态创建一个script元素，然后给script的scr属性赋值为ajax请求接口地址，在接口地址的后面添加一个callback的参数，参数的值就是jquery中动态生成的函数的函数名称，后台更具参数返回值

返回的结构是  方法名（json值）  成功后就相当于执行了一个js函数，就是ajax中success的成功后的回调函数

 8.前后端分离，API谁写的

和后台一般怎么沟通

Node面试题

**1.** 为什么要用**node?**

参考答案: 总结起来node有以下几个特点:简单强大，轻量可扩展．简单体现在node使用的是javascript,json来进行编码，人人都会；强大体现在非阻塞IO,可以适应分块传输数据，较慢的网络环境，尤其擅长高并发访问；轻量体现在node本身既是代码，又是服务器，前后端使用统一语言;可扩展体现在可以轻松应对多实例，多服务器架构，同时有海量的第三方应用组件．

**2. node**的构架是什么样子的**?**

参考答案: 主要分为三层，应用app >> V8及node内置架构 >> 操作系统. V8是node运行的环境，可以理解为node虚拟机．node内置架构又可分为三层: 核心模块(javascript实现) >> c++绑定 >> libuv + CAes + http.

**3. node**有哪些核心模块**?**

参考答案: EventEmitter, Stream, FS, Net和全局对象

node全局对象

**1. node**有哪些全局对象**?**

参考答案: process, console, Buffer和exports

**2. process**有哪些常用方法**?**

参考答案: process.stdin, process.stdout, process.stderr, process.on, process.env, process.argv, process.arch, process.platform, process.exit

**3. console**有哪些常用方法**?**

参考答案: console.log/console.info, console.error/console.warning, console.time/console.timeEnd, console.trace, console.table

**4. node**有哪些定时功能**?**

参考答案: setTimeout/clearTimeout, setInterval/clearInterval, setImmediate/clearImmediate, process.nextTick

node中的事件循环是什么样子的?

总体上执行顺序是：process.nextTick >> setImmidate >> setTimeout/SetInterval 看官网吧：https://github.com/nodejs/node/blob/master/doc/topics/event-loop-timers-and-nexttick.md

node中的Buffer如何应用?

参考答案: Buffer是用来处理二进制数据的，比如图片，mp3,数据库文件等.Buffer支持各种编码解码，二进制字符串互转．

**EventEmitter**

**1.** 什么是**EventEmitter?**

参考答案: EventEmitter是node中一个实现观察者模式的类，主要功能是监听和发射消息，用于处理多模块交互问题.

**2.** 如何实现一个**EventEmitter?**

参考答案: 主要分三步：定义一个子类，调用构造函数，继承EventEmitter

代码演示

    var util = require('util');

    var EventEmitter = require('events').EventEmitter;

    function MyEmitter() {

        EventEmitter.call(this);

    } // 构造函数

    util.inherits(MyEmitter, EventEmitter); // 继承

    var em = new MyEmitter();

    em.on('hello', function(data) {

        console.log('收到事件hello的数据:', data);

    }); // 接收事件，并打印到控制台

    em.emit('hello', 'EventEmitter传递消息真方便!');

**3. EventEmitter**有哪些典型应用**?**

参考答案: 1) 模块间传递消息 2) 回调函数内外传递消息 3) 处理流数据，因为流是在EventEmitter基础上实现的. 4) 观察者模式发射触发机制相关应用

**4.** 怎么捕获**EventEmitter**的错误事件**?**

参考答案: 监听error事件即可．如果有多个EventEmitter,也可以用domain来统一处理错误事件.

代码演示

    var domain = require('domain');

    var myDomain = domain.create();

    myDomain.on('error', function(err){

        console.log('domain接收到的错误事件:', err);

    }); // 接收事件并打印

    myDomain.run(function(){

        var emitter1 = new MyEmitter();

        emitter1.emit('error', '错误事件来自emitter1');

        emitter2 = new MyEmitter();

        emitter2.emit('error', '错误事件来自emitter2');

    });

**5. EventEmitter**中的**newListenser**事件有什么用处**?**

参考答案: newListener可以用来做事件机制的反射，特殊应用，事件管理等．当任何on事件添加到EventEmitter时，就会触发newListener事件，基于这种模式，我们可以做很多自定义处理.

代码演示

var emitter3 = new MyEmitter();

emitter3.on('newListener', function(name, listener) {

    console.log("新事件的名字:", name);

    console.log("新事件的代码:", listener);

    setTimeout(function(){ console.log("我是自定义延时处理机制"); }, 1000);

});

emitter3.on('hello', function(){

    console.log('hello　node');

});

Stream

**1.** 什么是**Stream?**

参考答案: stream是基于事件EventEmitter的数据管理模式．由各种不同的抽象接口组成，主要包括可写，可读，可读写，可转换等几种类型．

**2. Stream**有什么好处**?**

参考答案: 非阻塞式数据处理提升效率，片断处理节省内存，管道处理方便可扩展等.

**3. Stream**有哪些典型应用**?**

参考答案: 文件，网络，数据转换，音频视频等.

**4.** 怎么捕获**Stream**的错误事件**?**

参考答案: 监听error事件，方法同EventEmitter.

**5.** 有哪些常用**Stream,**分别什么时候使用**?**

参考答案: Readable为可被读流，在作为输入数据源时使用；Writable为可被写流,在作为输出源时使用；Duplex为可读写流,它作为输出源接受被写入，同时又作为输入源被后面的流读出．Transform机制和Duplex一样，都是双向流，区别时Transfrom只需要实现一个函数\_transfrom(chunk, encoding, callback);而Duplex需要分别实现\_read(size)函数和\_write(chunk, encoding, callback)函数.

**6.** 实现一个**Writable Stream?**

参考答案: 三步走:1)构造函数call Writable 2)　继承Writable 3) 实现\_write(chunk, encoding, callback)函数

代码演示

var Writable = require('stream').Writable;

var util = require('util');

function MyWritable(options) {

    Writable.call(this, options);

} // 构造函数

util.inherits(MyWritable, Writable); // 继承自Writable

MyWritable.prototype.\_write = function(chunk, encoding, callback) {

    console.log("被写入的数据是:", chunk.toString()); // 此处可对写入的数据进行处理

    callback();

};

process.stdin.pipe(new MyWritable()); // stdin作为输入源，MyWritable作为输出源

文件系统

**1.** 内置的**fs**模块架构是什么样子的**?**

参考答案: fs模块主要由下面几部分组成: 1) POSIX文件Wrapper,对应于操作系统的原生文件操作 2) 文件流 fs.createReadStream和fs.createWriteStream 3) 同步文件读写,fs.readFileSync和fs.writeFileSync 4) 异步文件读写, fs.readFile和fs.writeFile

**2.** 读写一个文件有多少种方法**?**

参考答案: 总体来说有四种: 1) POSIX式低层读写 2) 流式读写 3) 同步文件读写 4) 异步文件读写

**3.** 怎么读取**json**配置文件**?**

参考答案: 主要有两种方式，第一种是利用node内置的require('data.json')机制，直接得到js对象; 第二种是读入文件入内容，然后用JSON.parse(content)转换成js对象．二者的区别是require机制情况下，如果多个模块都加载了同一个json文件，那么其中一个改变了js对象，其它跟着改变，这是由node模块的缓存机制造成的，只有一个js模块对象; 第二种方式则可以随意改变加载后的js变量，而且各模块互不影响，因为他们都是独立的，是多个js对象.

**4. fs.watch**和**fs.watchFile**有什么区别，怎么应用?

参考答案: 二者主要用来监听文件变动．fs.watch利用操作系统原生机制来监听，可能不适用网络文件系统; fs.watchFile则是定期检查文件状态变更，适用于网络文件系统，但是相比fs.watch有些慢，因为不是实时机制．

网络

**1. node**的网络模块架构是什么样子的**?**

参考答案: node全面支持各种网络服务器和客户端，包括tcp, http/https, tcp, udp, dns, tls/ssl等.

**2. node**是怎样支持**https,tls**的**?**

参考答案: 主要实现以下几个步骤即可: 1) openssl生成公钥私钥 2) 服务器或客户端使用https替代http 3) 服务器或客户端加载公钥私钥证书

**3.** 实现一个简单的**http**服务器**?(**必须会手写**)**

参考答案: 经典又很没毛意义的一个题目．思路是加载http模块，创建服务器，监听端口.

代码演示

    var http = require('http'); // 加载http模块

    http.createServer(function(req, res) {

        res.writeHead(200, {'Content-Type': 'text/html'}); // 200代表状态成功, 文档类型是给浏览器识别用的

        res.write('<meta charset="UTF-8"><h1>我是标题啊！</h1><font color="red">这么原生，初级的服务器，下辈子能用着吗?!</font>'); // 返回给客户端的html数据

        res.end(); // 结束输出流

    }).listen(3000); // 绑定3ooo, 查看效果请访问 http://localhost:3000

child-process

**1.** 为什么需要**child-process?**

参考答案: node是异步非阻塞的，这对高并发非常有效．可是我们还有其它一些常用需求，比如和操作系统shell命令交互，调用可执行文件，创建子进程进行阻塞式访问或高CPU计算等，child-process就是为满足这些需求而生的．child-process顾名思义，就是把node阻塞的工作交给子进程去做．

**2. exec,execFile,spawn**和**fork**都是做什么用的**?**

参考答案: exec可以用操作系统原生的方式执行各种命令，如管道 cat ab.txt | grep hello; execFile是执行一个文件; spawn是流式和操作系统进行交互; fork是两个node程序(javascript)之间时行交互.

**3.** 实现一个简单的命令行交互程序**?**

参考答案: 那就用spawn吧.

代码演示

    var cp = require('child\_process');

    var child = cp.spawn('echo', ['你好', "钩子"]); // 执行命令

    child.stdout.pipe(process.stdout); // child.stdout是输入流，process.stdout是输出流

    // 这句的意思是将子进程的输出作为当前程序的输入流，然后重定向到当前程序的标准输出，即控制台

**4.** 两个**node**程序之间怎样交互**?**

参考答案: 用fork嘛，上面讲过了．原理是子程序用process.on, process.send，父程序里用child.on,child.send进行交互.

代码演示

    1) fork-parent.js

    var cp = require('child\_process');

    var child = cp.fork('./fork-child.js');

    child.on('message', function(msg){

        console.log('老爸从儿子接受到数据:', msg);

    });

    child.send('我是你爸爸，送关怀来了!');

    2) fork-child.js

    process.on('message', function(msg){

        console.log("儿子从老爸接收到的数据:", msg);

        process.send("我不要关怀，我要银民币！");

    });

**5.** 怎样让一个**js**文件变得像**linux**命令一样可执行?

参考答案: 1) 在myCommand.js文件头部加入#!/usr/bin/env node 2) chmod命令把js文件改为可执行即可 3) 进入文件目录，命令行输入myComand就是相当于node myComand.js了

**6. child-process**和**process**的**stdin,stdout,stderror**是一样的吗**?**

参考答案: 概念都是一样的，输入，输出，错误，都是流．区别是在父程序眼里，子程序的stdout是输入流，stdin是输出流．

**node**高级话题**(**异步，部署，性能调优，异常调试等**)**

**1. node**中的异步和同步怎么理解

参考答案: node是单线程的，异步是通过一次次的循环事件队列来实现的．同步则是说阻塞式的IO,这在高并发环境会是一个很大的性能问题，所以同步一般只在基础框架的启动时使用，用来加载配置文件，初始化程序什么的．

**2.** 有哪些方法可以进行异步流程的控制**?**

参考答案: 1) 多层嵌套回调 2)　为每一个回调写单独的函数，函数里边再回调3) 用第三方框架比方async, q, promise等

**3.** 怎样绑定**node**程序到**80**端口**?**

参考答案: 多种方式 1) sudo 2) apache/nginx代理 3) 用操作系统的firewall iptables进行端口重定向

**4.** 有哪些方法可以让**node**程序遇到错误后自动重启**?**

参考答案: 1) runit 2) forever 3) nohup npm start &

**5.** 怎样充分利用多个**CPU?**

参考答案: 一个CPU运行一个node实例

**6.** 怎样调节**node**执行单元的内存大小**?**

参考答案: 用--max-old-space-size 和 --max-new-space-size 来设置 v8 使用内存的上限

**7.** 程序总是崩溃，怎样找出问题在哪里**?**

参考答案: 1) node --prof 查看哪些函数调用次数多 2) memwatch和heapdump获得内存快照进行对比，查找内存溢出

**8.** 有哪些常用方法可以防止程序崩溃?

参考答案: 1) try-catch-finally 2) EventEmitter/Stream error事件处理 3) domain统一控制 4) jshint静态检查 5) jasmine/mocha进行单元测试

**9.** 怎样调试**node**程序**?**

参考答案: node --debug app.js 和node-inspector

常用知名第三方类库**(Async, Express**等)

**1. async**都有哪些常用方法，分别是怎么用**?**

参考答案: async是一个js类库，它的目的是解决js中异常流程难以控制的问题．async不仅适用在node.js里，浏览器中也可以使用． 1) async.parallel并行执行完多个函数后，调用结束函数

    async.parallel([

        function(){ ... },

        function(){ ... }

    ], callback);

**2) async.series**串行执行完多个函数后，调用结束函数

    async.series([

        function(){ ... },

        function(){ ... }

    ]);

**3) async.waterfall**依次执行多个函数，后一个函数以前面函数的结果作为输入参数

    async.waterfall([

        function(callback) {

            callback(null, 'one', 'two');

        },

        function(arg1, arg2, callback) {

          // arg1 now equals 'one' and arg2 now equals 'two'

            callback(null, 'three');

        },

        function(arg1, callback) {

            // arg1 now equals 'three'

            callback(null, 'done');

        }

    ], function (err, result) {

        // result now equals 'done'

    });

**4) async.map**异步执行多个数组，返回结果数组

    async.map(['file1','file2','file3'], fs.stat, function(err, results){

        // results is now an array of stats for each file

    });

**5) async.filter**异步过滤多个数组，返回结果数组

    async.filter(['file1','file2','file3'], fs.exists, function(results){

        // results now equals an array of the existing files

    });

**2. express**项目的目录大致是什么样子的

参考答案: app.js, package.json, bin/www, public, routes, views.

**3. express**常用函数

参考答案: express.Router路由组件,app.get路由定向，app.configure配置，app.set设定参数,app.use使用中间件

**4. express**中如何获取路由的参数

参考答案: /users/:name使用req.params.name来获取; req.body.username则是获得表单传入参数username; express路由支持常用通配符 ?, +, \*, and ()

**5. express response**有哪些常用方法

参考答案: res.download() 弹出文件下载

res.end() 结束response

res.json() 返回json

res.jsonp() 返回jsonp

res.redirect() 重定向请求

res.render() 渲染模板

res.send() 返回多种形式数据

res.sendFile 返回文件

res.sendStatus() 返回状态

其它相关后端常用技术(MongoDB, Redis, Apache, Nginx等)

**1. mongodb**有哪些常用优化措施

参考答案: 类似传统数据库，索引和分区．

**2. mongoose**是什么？有支持哪些特性**?**

参考答案: mongoose是mongodb的文档映射模型．主要由Schema, Model和Instance三个方面组成．Schema就是定义数据类型，Model就是把Schema和js类绑定到一起，Instance就是一个对象实例．常见mongoose操作有,save, update, find. findOne, findById, static方法等．

**2. redis**支持哪些功能

参考答案: set/get, mset/hset/hmset/hmget/hgetall/hkeys, sadd/smembers, publish/subscribe, expire

**3. redis**最简单的应用

参考答案:

    var redis = require("redis"),

        client = redis.createClient();

    client.set("foo\_rand000000000000", "some fantastic value");

    client.get("foo\_rand000000000000", function (err, reply) {

        console.log(reply.toString());

    });

    client.end();

**4. apache,nginx**有什么区别**?**

参考答案: 二者都是代理服务器，功能类似．apache应用简单，相当广泛．nginx在分布式，静态转发方面比较有优势．

工具篇面试题

 1、svn git上传合并项目怎么做？

先用git初始化界面 项目路径 j加上分知名

然后git status 查看状态，

在就是跟踪文件 git add

在分支上提交 git commit -m “message

切换分支 git checkout branch name

合并分支 git merfe branch\_name

更新master Git svn rebase

然后 将代码提交到svn上git SVN dcommit

2、webpack热加载的原理

*构建 bundle 的时候，加入一段 HMR runtime 的 js 和一段和服务沟通的 js 。文件修改会触发 webpack 重新构建，服务器通过向浏览器发送更新消息，浏览器通过 jsonp 拉取更新的模块文件，jsonp 回调触发模块热替换逻辑。*

3、webpack 打包按需加载

4、怎么确保代码质量？

*注释要写全，代码尽量精简，class命名是大驼峰式的命名，开头字母大写，xml文件里的命名都要小写，我想到的大概就这些，对了，还有包的命名要分类，结构要清晰。*

5、gulp是怎么合并文件的

*gulp 的 concat 插件主要用于合并文件，把也就是多个文件合并到一个文件中，以节省存储空间。
安装命令：cnpm install --save-dev gulp-conc*at

*gulpfile.js 文件配置：*

*var concat = require('gulp-concat');
gulp.task("scripts",function(){
   gulp.src("src/js/\*.js")
   .pipe(concat("all.js"))
   .pipe(gulp.dest("dist"))
});
在cmd 命令行控制器中执行定义好的任务：
gul*p concat

6、移动端怎么测试的

最简单的就是发到qq上打开就行。

7、抓包工具用过吧

*开始就用SpyNet，但是我一般都是直接再浏览器上自带的抓包工具。*

8、平时用git的话有什么优点缺点

比svn更快速的切换分支，因为分支都在本地。

可以随时提交。

缺点：

*上层封装和底层实现的差异太大*

*分支合并的历史很容易搞乱*

*各个版本之间的命令也有些许不兼容*

9、遇到冲突是怎么解决的？

1. glup 功能？

文件合并

压缩文件

语法检查

监听文件变化

11、gulp做压缩还是打包

12、简单介绍下webpack的工作原理

13、webpack的基本配置有哪些

*entry:打包的入口文件，一个字符串或者一个对象
　　　　output:配置打包的结果，对象
　　　　fileName：定义输出文件名，字符串
　　　　path：定义输出文件路径，字符串
　　　　module:定义对模块的处理逻辑，对象
　　　　loaders：定义一系列的加*载器，数组

1. grunt, gulp, webpack三者的区别

*Gulp / Grunt 可以理解为帮助****前端自动化的工具****,旨在提供一个自动化的流程（less，stylus，sass已经babel的转码，图片的压缩，代码的压缩复制等系列操作)*

*webpack 提供的是一个****前端模块化的方案****,让我们可以将commonJS的模块方式应用与浏览器端;*

15、webpack工具和node了解哪些?node各个模块的底层原理?

16、webpack打包是如何处理css图片的

17、用node发过包吗，步骤是什么

18、webpack打包的配置，常见的loaders和plugins？

19、版本的管理工具，是用通讯化工具，还是直接敲的命令行

20、grunt和gulp？

21、webpack和gulp项目里你是怎么用的，

项目类面试题

1，自己封装过组件吗？怎么封装？

封装过，

2，页面效果是自己写还是用插件？

3，项目是否是自己独立完成的

4，工作中对一些框架等有什么要求

5，会一个人做一个项目，还是一起做

6，项目都是你们前端做的么

7，公司是只做前端项目，还是前后端都在做

8，以前公司所用框架。

9，外包公司中你是否被外派做过什么项目。

10，项目周期时间中中途是否有变动。

11、工作时的强度

12、项目周期是多少

13、项目的大小人员分派会有所改变吗？

14、自己接触过的大项目是什么 大概开发多久

15、在外包期间可以考虑运用新框架或申请吗

16、上一家公司主要是做什么的？

17、对自己的学习能力如何？

18、在用vue做项目时遇到了什么问题？

19、怎么区别WebApp和App？

20、上家公司是什么公司？用的哪个框架居多

21、外包公司都外派到哪里？

22、地图有没有做过？怎么进行的定位？怎么确定某一点的坐标？

23、做移动端项目有没有调过手机的图库、照相机之类的？怎么调用的？

24、购物App用户头像怎么上传？

25、如果做一个企业的网站，包括移动端和PC端，你一个人需要多久？

26、以前公司是UI切图还是自己切图

27、你对公司做的项目有什么要求吗？（比如那些类型的项目能做，那些不能做）

28、上家公司你主要负责什么

29、移动端的页面写的多吗

30、做的单页网页多吗

31、写移动端页面一般用什么布局

32、以后前端的方向有什么规划

33、说你爱上了前端，你爱前端的什么东西

34、之前工作的流程是什么样的，包括需求，评审，开发，上线流程

35、上一家公司用的是requires还是seats

36、这有一个多语言要是前端来实现，你想怎么实现

37、工作之中有没有碰到什么难点不好解决的问题

39、如果说实现不了或实现有困难，那应该怎么做

40、就是已经评估完成，确定要几天上线，最后却发现做不了，要是你你会怎么办

41、如果定死了不能改，搞不定这个项目，必须上线怎么办？

42、你认为哪些网站适合做响应式，哪些网站不适合？

43、你平时研究那些前沿的技术吗

44、后台管理系统有没有做过

45、java有没有了解一点

46、echarts框架 数据图表可视化

47、less 怎么用编译

48、负载均衡你了解吗？阐述一下

49、linux环境你熟悉吗？说一些你用过的指令

50、为什么选择做前端（

51、你希望进入一个什么样的团队

52、你有什么问题想问我（面试官）的吗？

53、你掌握那些前端技术？

54、你上家公司在做移动端时，开发流程是什么？

55、介绍一下你做的这个项目，进一步细问：整个项目有哪些模块，你主要负责哪些

56、你在项目中的角色

57、你在项目中做的最出彩的一个地方

58、碰到过什么样的困难，怎么解决的

59、（如果你是这个项目的负责人），任务怎么分配的，有没有关注过团队成员的成长问题

60、介绍一下自己的项目经历，可以看着自己的简历介绍(用到的技术 + 遇到的技术难点)；

61、.webapp你主要做的是哪个模块，是页面布局做的多还是交互做的多

有一个a页面和一个b页面都是一个网站中的两个不同的页面怎么实现两个页面直接的数据通信？

公司有哪些部门，团队组成？

购物车功能是怎么实现的，添加购物车是只要前端操作就行吗？

如何提升列表页的渲染速度，模板引擎用过哪些？

在应用第三方插件时，发现与自己的项目功能不是百分百匹配，如何解决？

进行技术选型时，都考虑哪些因素

登录过后如何保持登录状态

购物车的流程（前后端结合）是什么

俩数组第一个数组有俩个对象，其中一个对象里面的键值对分别是name和age，你吧这个数组里面name和age重复的去一下重，放到第二个数组里面

如果一个店长拥有比较高的权限，可以看到下面的员工甲和员工乙不同的权限，虽然它俩是同级的，但店长给他分配了不同的权限，所以看到的东西是不一样的。数据和按钮以及菜单是不一样的，这个大概的思路前端该怎么判断，需要哪些数据去判断？

账号，密码，点击登录，点击登录的时候需要请求一个数据，请求成功以后需要判断，如果所填的东西和后台账号密码的对得上，在sidesiaze里跳转，如果失败了是不可以跳转的。情况来了，如果提前在路由里配置好的，不管成功不成功点击都会跳页面。但是目前是如果请求失败了不让它跳，这个意思就是得在成功回调里用js跳转，不用localtion.href,有别的方法吗？

做购物车功能，如果用户未登录，但是点击了加入购物车，到结算的时候提示用户没有登录，那么现在如何实现，当用户登录成功后同步未登录状态下购物车内的数据？

熟悉并应用过哪些第三方插件？

登录过后如何保持登录状态

前端研究篇

说一下你对前端工程化的看法吧

我们要做一个校园OA系统适合选用什么样的技术呀