目录

一 HTML

1.H5 新特性

1.H5 的新特性有哪些

1.画布(Canvas) API

2.地理(Geolocation) API

3.音频、视频 API(audio,video)

4.localStorage 和 sessionStorage

5.webworker, websocket

6.header,nav,footer,aside,article,section

7.web worker 是运行在浏览器后台的 js 程序，他不影响主程序的运行，是另开的一个 js 线程，可以用这个线程执行复杂的数据操作，然后把操作结果通过 postMessage 传递给主线程，这样在进行复杂且耗时的操作时就不会阻塞主线程了。

2.两种实现前端路由的方式

HTML5 History 两个新增的 API：history.pushState 和 history.replaceState，两个 API 都会操作浏览器的历史记录，而不会引起页面的刷新。

Hash 就是 url 中看到 # ,我们需要一个根据监听哈希变化触发的事件（hashchange）事件。我们用 window.location 处理哈希的改变时不会重新渲染页面，而是当作新页面加到历史记录中，这样我们跳转页面就可以在 hashchange 事件中注册 ajax 从而改变页面内容。 可以为 hash 的改变添加监听事件：

window.addEventListener("hashchange", funcRef, false)

优点

从性能和用户体验的层面来比较的话，后端路由每次访问一个新页面的时候都要向服务器发送请求，然后服务器再响应请求，这个过程肯定会有延迟。而前端路由在访问一个新页面的时候仅仅是变换了一下路径而已，没有了网络延迟，对于用户体验来说会有相当大的提升。

前端路由的优点有很多，比如页面持久性，像大部分音乐网站，你都可以在

播放歌曲的同时，跳转到别的页面而音乐没有中断，再比如前后端彻底分离。开发一个前端路由，主要考虑到页面的可插拔、页面的生命周期、内存管理等。

缺点

使用浏览器的前进，后退键的时候会重新发送请求，没有合理地利用缓存。

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：

window.history.pushState(stateObject, title, URL)

window.history.replaceState(stateObject, title, URL)

3.websocket

WebSocket 使用 ws 或 wss 协议，Websocket 是一个持久化的协议，相对于 HTTP 这种非持久的协议来说。WebSocket API 最伟大之处在于服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息。WebSocket 并不限于以 Ajax(或 XHR)方式通信，因为 Ajax 技术需要客户端发起请求，而 WebSocket 服务器和客户端可以彼此相互推送信息；XHR 受到域的限制，而 WebSocket 允许跨域通信。

```
// 创建一个 Socket 实例

var socket = new WebSocket('ws://localhost:8080');

// 打开 Socket

socket.onopen = function(event) {
```

```
    // 发送一个初始化消息
    socket.send('I am the client and I\'m listening!');
    // 监听消息
    socket.onmessage = function(event) {
      console.log('Client received a message',event);
    };
    // 监听 Socket 的关闭
    socket.onclose = function(event) {
      console.log('Client notified socket has closed',event);
    };
    // 关闭 Socket....
    //socket.close()
  };
```

4.webSocket 如何兼容低浏览器？(阿里)

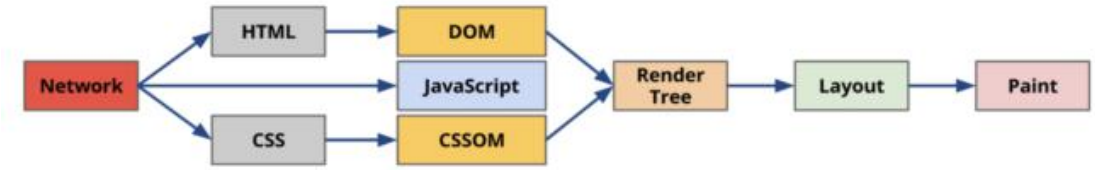
1.Adobe Flash Socket 、

2.ActiveX HTMLFile (IE) 、

3. 基于 multipart 编码发送 XHR 、

4.基于长轮询的 XHR

2.浏览器渲染原理解析



1、首先渲染引擎下载 HTML，解析生成 DOM Tree

2、遇到 css 标签或 JS 脚本标签就新起线程去下载他们，并继续构建 DOM。（其中 css 是异步下载同步执行）浏览器引擎通过 DOM Tree 和 CSS Rule Tree 构建 Rendering Tree

3、 通过 CSS Rule Tree 匹配 DOM Tree 进行定位坐标和大小，这个过程称为 Flow 或 Layout 。

4、最终通过调用 Native GUI 的 API 绘制网页画面的过程称为 Paint 。

当用户在浏览网页时进行交互或通过 js 脚本改变页面结构时，以上的部分操作有可能重复运行，此过程称为 Repaint 或 Reflow。 重排是指 dom 树发生结构变化后，需要重新构建 dom 结构。 重绘是指 dom 节点样式改变，重新绘制。 重排一定会带来重绘，重绘不一定有重排。

如何减少浏览器重排：将需要多次重排的元素，position 属性设为 absolute 或 fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。

3.iframe 有那些缺点?

1. iframe 会阻塞主页面的 Onload 事件;

2. 搜索引擎的检索程序无法解读这种页面，不利于 SEO;

3. 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。

4. 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

5. 动态给 iframe 添加 src 属性值，这样可以绕开以上两个问题。

4.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>

5.HTML5 的 form 如何关闭自动完成功能?

给不想要提示的 form 或某个 input 设置为 autocomplete=off。

6.如何实现浏览器内多个标签页之间的通信? (阿里)

1. WebSocket、SharedWorker;

2. 也可以调用 localstorge、cookies 等本地存储方式;

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;

注意 quirks：Safari 在无痕模式下设置 localstorge 值时会抛出 QuotaExceededError 的异常;

7.页面可见性（Page Visibility API） 可以有哪些用途?

1 .通过 visibilityState 的值检测页面当前是否可见，以及打开网页的时间等;

2. 在页面被切换到其他后台进程的时候，自动暂停音乐或视频的播放;

8.实现不使用 border 画出 1px 高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果。

9.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：会重读，而<B>是展示强调内容。

i 内容展示为斜体，me 表示强调的文本；

Physical Style Elements -- 自然样式标签

b, i, u, s, pre

Semantic Style Elements -- 语义样式标签

strong, em, ins, del, code

应该准确使用语义样式标签，但不能滥用，如果不能确定时首选使用自然样式标签。

10.你做的页面在哪些流览器测试过？这些浏览器的内核分别是什么？

Trident 内核:IE 系列

Gecko 内核:Firefox

Webkit 内核:Safari

Blink 内核：是基于 Webkit 内核的子项目,使用的浏览器有：

Chrome/opera 等除 IE、Firefox、Safari 之外的几乎所有浏览器

几乎所有国产双内核浏览器（Trident/Blink）如 360、猎豹、qq、百度等

11.每个 HTML 文件里开头都有个很重要的东西，Doctype，知道这是干什么的吗?

文档声明。

<!DOCTYPE> 声明位于文档中的最前面的位置，处于 <html> 标签之前。此标签可告知浏览器文档使用哪种 HTML 或 XHTML 规范。（重点：告诉浏览器按照何种规范解析页面）

IE 下如不书写文档声明会使用怪异模式解析网页导致一系列 CSS 兼容性问题。

12.div+css 的布局较 table 布局有什么优点?

正常场景一般都适用 div+CSS 布局，

优点：

1. 结构与样式分离
2. 代码语义性好
3. 更符合 HTML 标准规范
4. SEO 友好

Table 布局的适用场景：

某种原因不方便加载外部 CSS 的场景，例如邮件正文，此时用 table 布局可以在无 css 情况下保持页面布局正常。

13.img 的 alt 与 title 有何异同? strong 与 em 的异同?

a:alt(alt text):为不能显示图像、窗体或 applets 的用户代理（UA），alt 属性用来指定替换文字。替换文字的语言由 lang 属性指定。(在 IE 浏览器下会在没有 title 时把 alt 当成 tool tip 显示)

title(tool tip):该属性为设置该属性的元素提供建议性的信息。

em:表现为斜体，语义表示强调

strong:表现为粗体，语义为强烈语气，强调程度超过 em

14.简述一下 src 与 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，例如 js 脚本，img 图片和 frame 等元素。

<script src =”js.js”></script>

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。

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。

15.知道的网页制作会用到的图片格式有哪些?

png-8，png-24，jpeg，gif，svg。

但是上面的那些都不是面试官想要的最后答案。面试官希望听到是 Webp。(是否有关注新技术，新鲜事物)

科普一下 Webp：WebP 格式，谷歌（google）开发的一种旨在加快图片加载速度的图片格式。图片压缩体积大约只有 JPEG 的 2/3，并能节省大量的服务器带宽资源和数据空间。Facebook Ebay 等知名网站已经开始测试并使用 WebP 格式。

在质量相同的情况下，WebP 格式图像的体积要比 JPEG 格式图像小 40%

16.在 css/js 代码上线之后开发人员经常会优化性能，从用户刷新网页开始，一次 js 请求一般情况下有哪些地方会有缓存处理?

dns 缓存，cdn 缓存，浏览器缓存，服务器缓存。

17.一个页面上有大量的图片（大型电商网站），加载很慢，你有哪些方法优化这些图片的加载，给用户更好的体验。

图片懒加载，在页面上的未可视区域可以添加一个滚动条事件，判断图片位置与浏览器顶端的距离与页面的距离，如果前者小于后者，优先加载。

如果为幻灯片、相册等，可以使用图片预加载技术，将当前展示图片的前一张和后一张优先下载。

如果图片为 css 图片，可以使用 CSSsprite，SVGsprite，Iconfont、Base64 等技术。

如果图片过大，可以使用特殊编码的图片，加载时会先加载一张压缩的特别厉害的缩略图，以提高用户体验。

如果图片展示区域小于图片的真实大小，则因在服务器端根据业务需要先行进行图片压缩，图片压缩后大小与展示一致。

18.你如何理解 HTML 结构的语义化?

HTML 结构语义化：

更符合 W3C 统一的规范标准，是技术趋势。

没有样式时浏览器的默认样式也能让页面结构很清晰。

对功能障碍用户友好。屏幕阅读器（如果访客有视障）会完全根据你的标记来“读”你的网页。

对其他非主流终端设备友好。例如机顶盒、PDA、各种移动终端。

对 SEO 友好。

19.谈谈以前端角度出发做好 SEO 需要考虑什么?

搜索引擎主要以:

外链数量和质量

网页内容和结构

来决定某关键字下的网页搜索排名。

前端应该注意网页结构和内容方面的情况:

Meta 标签优化

主要包括主题（Title），网站描述(Description)。还有一些其它的隐藏文字比如 Author（作者），Category（目录），Language（编码语种）等。

符合 W3C 规范的语义性标签的使用。

如何选取关键词并在网页中放置关键词

搜索就得用关键词。关键词分析和选择是 SEO 最重要的工作之一。首先要给网站确定主关键词（一般在 5 个上下），然后针对这些关键词进行优化，包括关键词密度（Density），相关度（Relavancy），突出性（Prominency）等等。

20.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?

新特性:

1. 拖拽释放(Drag and drop) API

2. 语义化更好的内容标签（header,nav,footer,aside,article,section）

3. 音频、视频 API(audio,video)

4. 画布(Canvas) API

5. 地理(Geolocation) API

6. 本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；

7. sessionStorage 的数据在浏览器关闭后自动删除

8. 表单控件，calendar、date、time、email、url、search

9. 新的技术 webworker, websocket, Geolocation

移除的元素：

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

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

支持 HTML5 新标签：

1. 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（当然最好的方式是直接使用成熟的框架、使用最多的是 html5shim 框架）：

<!--[if lt IE 9]>

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

<![endif]-->

如何区分：

DOCTYPE 声明新增的结构元素、功能元素

21.HTML5 Canvas 元素有什么用?

Canvas 元素用于在网页上绘制图形，该元素标签强大之处在于可以直接在 HTML 上进行图形操作。

22.如何在 HTML5 页面中嵌入音频?

HTML 5 包含嵌入音频文件的标准方式，支持的格式包括 MP3、Wav 和 Ogg：

```
<audio controls>
  <source src="jamshed.mp3" type="audio/mpeg">
   Your browser does'nt support audio embedding feature.
</audio>
```

23.如何在 HTML5 页面中嵌入视频?

和音频一样，HTML5 定义了嵌入视频的标准方法，支持的格式包括：MP4、WebM 和 Ogg：

```
<video width="450" height="340" controls>
  <source src="jamshed.mp4" type="video/mp4">
   Your browser does'nt support video embedding feature.
</video>
```

24.HTML5 引入什么新的表单属性?

Datalist datetime output keygen date month week time number range emailurl

二 CSS

1.box-sizing

box-sizing 属性主要用来控制元素的盒模型的解析模式。默认值是 content-box。

content-box：让元素维持 W3C 的标准盒模型。元素的宽度/高度由 border + padding + content 的宽度/高度决定，设置 width/height 属性指的是 content 部分的宽/高

border-box：让元素维持 IE 传统盒模型（IE6 以下版本和 IE6~7 的怪异模式）。设置 width/height 属性指的是 border + padding + content

应用场景：统一风格的表单元素 表单中有一些 input 元素其实还是展现的是传统 IE 盒模型，带有一些默认的样式，而且在不同平台或者浏览器下的表现不一，造成了表单展现的差异。此时我们可以通过 box-sizing 属性来构建一个风格统一的表单元素。

2.水平垂直居中的方法

行内布局

line-height + text-align vertical-align + text-align

块布局

position absolute + margin auto position absolute + negative margin position absolute + translate(-50%, -50%)

父容器子容器不确定宽高的块级元素，做上下居中

1.flex

```
#wrap{
      display:flex;
      justify-content:center;
      align-items:center;
}
```

2.tabel

```
.parent {
    text-align: center;//水平居中
    display: table-cell;
    vertical-align: middle;//垂直居中
}
.child {
      display: inline-block;//防止块级元素宽度独占一行，内联元素可不设置
}
```

3.absolute+transform 水平垂直居中

```
<div class="parent">
   <div class="child">Demo</div>
```

```
</div>


<style>
   .parent {
      position: relative;
   }
   .child {
      position: absolute;
      left: 50%;
      top: 50%;
      transform: translate(-50%, -50%);
   }
</style>
```

4.webkit-box

```
//对父级元素设置
position: relative;
display: -webkit-box;
-webkit-box-align: center;
-webkit-box-pack: center;
```

for detail: https://github.com/hawx1993/tech-blog/issues/12

3.实现左边定宽右边自适应效果

1.table(父级元素)与 tabel-cell（两个子集元素）

2.flex(父级元素)+flex :1（右边子元素）

3.左边定宽，并且左浮动；右边设置距离左边的宽度

4.左边定宽，左边设置 position:absolute；右边设置距离左边的宽度

4.三列布局（中间固定两边自适应宽度）

1. 采用浮动布局（左边左浮动，右边右浮动，中间 margin：0 宽度值）

2. 绝对定位方式（左右绝对定位，左边 left0 右边 right0，中间上同）

5.BFC（Block Formatting Contexts）块级格式化上下文

块格式化上下文（block formatting context） 是页面上的一个独立的渲染区域，容器里面的子元素不会在布局上影响到外面的元素。它是决定块盒子的布局及浮动元素相互影响的一个因素。

下列情况将创建一个块格式化上下文：

① float

② overflow

③ display（display 为 inline-block、table-cell）

④ position（absolute 或 fixed）

BFC 的作用

1.清除内部浮动：对子元素设置浮动后，父元素会发生高度塌陷，也就是父元素的高度变为 0。解决这个问题，只需要把把父元素变成一个 BFC 就行了。常用的办法是给父元素设置 overflow:hidden。

2.上下 margin 重合问题，可以通过触发 BFC 来解决

6.清除浮动元素的方法和各自的优缺点

清除浮动，实际上是清除父元素的高度塌陷。因为子元素脱离了父元素的文档流，所以，父元素失去了高度，导致了塌陷。要解决这个问题，就是让父元素具有高度。

浮动元素的特性： 在正常布局中位于该浮动元素之下的内容，此时会围绕着浮动元素，填满其右侧的空间。浮动到右侧的元素，其他内容将从左侧环绕它（浮动元素影响的不仅是自己，它会影响周围的元素对其进行环绕。float 仍会占据其位置，position:absolute 不占用页面空间 会有重叠问题 ）

1.在浮动元素末尾添加空标签清除浮动 clear:both （缺点：增加无意义标签）

2.给父元素设置 overflow:auto 属性

3.after 伪元素

7.css 实现自适应正方形

方案一：CSS3 vw 单位

方案二：设置垂直方向的 padding 撑开容器

方案三：利用伪元素的 margin(padding)-top 撑开容器

8.position 的值

1. absolute :生成绝对定位的元素， 相对于最近一级的 定位不是 static 的父元素来进行定位。

2. fixed（老 IE 不支持)生成绝对定位的元素,通常相对于浏览器窗口或 frame 进行定位。

3. relative 生成相对定位的元素，相对于其在普通流中的位置进行定位。

4. static 默认值。没有定位，元素出现在正常的流中

5. sticky 生成粘性定位的元素，容器的位置根据正常文档流计算得出

9.如何在页面上实现一个圆形的可点击区域?

1、map+area 或者 svg

2、border-radius

3、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

10.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?

（1）有两种， IE 盒子模型、W3C 盒子模型；

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

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

11.CSS 选择符有哪些？哪些属性可以继承？

* 一、基本选择器

1. * 通用元素选择器，匹配任何元素

2. E 标签选择器，匹配所有使用 E 标签的元素

3. .info class 选择器，匹配所有 class 属性中包含 info 的元素

4. #footer id 选择器，匹配所有 id 属性等于 footer 的元素

二、多元素的组合选择器

5. E,F 多元素选择器，同时匹配所有 E 元素或 F 元素，E 和 F 之间用逗号分隔

6. E F 后代元素选择器，匹配所有属于 E 元素后代的 F 元素，E 和 F 之间用空格分隔

7. E > F 子元素选择器，匹配所有 E 元素的子元素 F

8. E + F 毗邻元素选择器，匹配所有紧随 E 元素之后的同级元素 F

三、CSS 2.1 属性选择器

9. E[att] 匹配所有具有 att 属性的 E 元素，不考虑它的值。（注意：E 在此处可以省略，比如"[cheacked]"。以下同。）

10. E[att=val] 匹配所有 att 属性等于"val"的 E 元素

11. E[att~=val] 匹配所有 att 属性具有多个空格分隔的值、其中一个值等于"val"的 E 元素

12. E[att|=val] 匹配所有 att 属性具有多个连字号分隔

（hyphen-separated）的值、其中一个值以"val"开头的 E 元素，主要用于 lang 属性，比如"en"、"en-us"、"en-gb"等等

四、CSS 2.1 中的伪类

13. E:first-child 匹配父元素的第一个子元素

14. E:link 匹配所有未被点击的链接

15. E:visited 匹配所有已被点击的链接

16. E:active 匹配鼠标已经其上按下、还没有释放的 E 元素

17. E:hover 匹配鼠标悬停其上的 E 元素

18. E:focus 匹配获得当前焦点的 E 元素

19. E:lang(c) 匹配 lang 属性等于 c 的 E 元素

五、 CSS 2.1 中的伪元素

20. E:first-line 匹配 E 元素的第一行

21. E:first-letter 匹配 E 元素的第一个字母

22. E:before 在 E 元素之前插入生成的内容

23. E:after 在 E 元素之后插入生成的内容

六、CSS 3 的同级元素通用选择器

24. E ~ F 匹配任何在 E 元素之后的同级 F 元素

七、CSS 3 属性选择器

25. E[att^="val"] 属性 att 的值以"val"开头的元素

26. E[att$="val"] 属性 att 的值以"val"结尾的元素

27. E[att*="val"] 属性 att 的值包含"val"字符串的元素

八、CSS 3 中与用户界面有关的伪类

28. E:enabled 匹配表单中激活的元素

29. E:disabled 匹配表单中禁用的元素

30. E:checked 匹配表单中被选中的 radio（单选框）或 checkbox（复选框）元素

31. E::selection 匹配用户当前选中的元素

九、CSS 3 中的结构性伪类

32. E:root 匹配文档的根元素，对于 HTML 文档，就是 HTML 元素

33. E:nth-child(n) 匹配其父元素的第 n 个子元素，第一个编号为 1

34. E:nth-last-child(n) 匹配其父元素的倒数第 n 个子元素，第一个编号为 1

35. E:nth-of-type(n) 与:nth-child()作用类似，但是仅匹配使用同种标签的元素

36. E:nth-last-of-type(n) 与:nth-last-child() 作用类似，但是仅匹配使用同种标签的元素

37. E:last-child 匹配父元素的最后一个子元素，等同

于:nth-last-child(1)

38. E:first-of-type 匹配父元素下使用同种标签的第一个子元素，等同于:nth-of-type(1)

39. E:last-of-type 匹配父元素下使用同种标签的最后一个子元素，等同于:nth-last-of-type(1)

40. E:only-child 匹配父元素下仅有的一个子元素，等同于:first-child:last-child 或 :nth-child(1):nth-last-child(1)

41. E:only-of-type 匹配父元素下使用同种标签的唯一一个子元素，等同于:first-of-type:last-of-type 或 :nth-of-type(1):nth-last-of-type(1)

42. E:empty 匹配一个不包含任何子元素的元素，注意，文本节点也被看作子元素

十、CSS 3 的反选伪类

43. E:not(s) 匹配不符合当前选择器的任何元素

十一、CSS 3 中的 :target 伪类

44. E:target 匹配文档中特定"id"点击后的效果

* 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 ;

12.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:

同权重: 内联样式表（标签内部）> 嵌入样式表（当前文件中）> 外部样式表（外部文件中）。

!important > id > class > tag

important 比 内联优先级高

13.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。

14.display 有哪些值？说明他们的作用。

block 块类型。默认宽度为父元素宽度，可设置宽高，换行显示。

none 缺省值。象行内元素类型一样显示。

inline 行内元素类型。默认宽度为内容宽度，不可设置宽高，同行显示。

inline-block 默认宽度为内容宽度，可以设置宽高，同行显示。

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。

15.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）

缩放,定位,倾斜,动画,多背景

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

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

一个用于页面布局的全新 CSS3 功能，Flexbox 可以把列表放在同一个方向（从上到下排列，从左到右），并让列表能延伸到占用可用的空间。

较为复杂的布局还可以通过嵌套一个伸缩容器（flex container）来实现。

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。

它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"。

常规布局是基于块和内联流方向，而 Flex 布局是基于 flex-flow 流可以很方便的用来做局中，能对不同屏幕大小自适应。

在布局上有了比以前更加灵活的空间。

具体：http://www.w3cplus.com/css3/flexbox-basics.html

17.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;
}
```

18.一个满屏 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

19.css 多列等高如何实现?

利用 padding-bottom|margin-bottom 正负值相抵;

设置父容器设置超出隐藏(overflow:hidden),这样子父容器的高度就还是它里面的列没有设定 padding-bottom 时的高度,

当它里面的任 一列高度增加了,则父容器的高度被撑到里面最高那列的高度,

其他比这列矮的列会用它们的 padding-bottom 补偿这部分高度差。

20.经常遇到的浏览器的兼容性有哪些?原因,解决方法是什么,常用 hack 的技巧 ?

* png24 位的图片在 iE6 浏览器上出现背景,解决方案是做成 PNG8.

* 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。

* 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:red;/*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*/
}
```

* IE 下,可以使用获取常规属性的方法来获取自定义属性,

  也可以使用 getAttribute()获取自定义属性;

  Firefox 下,只能使用 getAttribute()获取自定义属性。

  解决方法:统一通过 getAttribute()获取自定义属性。

* 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。

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

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

超链接访问过后 hover 样式就不出现了 被点击访问过的超链接样式不在具有 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 {}

21.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。

22.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; }
```

23.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

24.CSS 里的 visibility 属性有个 collapse 属性值是干嘛用的？在不同浏览器下以后什么区别?

对于普通元素 visibility:collapse;会将元素完全隐藏,不占据页面布局空间,与 display:none;表现相同. 如果目标元素为 table,visibility:collapse;将 table 隐藏,但是会占据页面布局空间. 仅在 Firefox 下起作用,IE 会显示元素,Chrome 会将元素隐藏,但是占据空间.

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

如果元素的 display 为 none,那么元素不被渲染,position,float 不起作用,如果元素拥有 position:absolute;或者 position:fixed;属性那么元素将为绝对定位,float 不起作用.如果元素 float 属性不是 none,元素会脱离文档流,根据 float 属性值来显示.有浮动,绝对定位,inline-block 属性的元素,margin 不会和垂直方向上的其他元素 margin 折叠.

26.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。

27. 请解释一下为什么需要清除浮动？清除浮动的方式

清除浮动是为了清除使用浮动元素产生的影响。浮动的元素，高度会塌陷，而高度的塌陷使我们页面后面的布局不能正常显示。

1、父级 div 定义 height；

2、父级 div 也一起浮动；

3、常规的使用一个 class；

```
.clearfix::before, .clearfix::after {
        content: " ";
        display: table;
}
.clearfix::after {
        clear: both;
}
.clearfix {
        *zoom: 1;
}
```

4、SASS 编译的时候，浮动元素的父级 div 定义伪类:after

```
&::after,&::before{
        content: " ";
          visibility: hidden;
          display: block;
```

```
    height: 0;

    clear: both;

}
```

解析原理：

1) display:block 使生成的元素以块级元素显示,占满剩余空间;

2) height:0 避免生成内容破坏原有布局的高度。

3) visibility:hidden 使生成的内容不可见，并允许可能被生成内容盖住的内容可以进行点击和交互;

4）通过 content:"."生成内容作为最后一个元素，至于 content 里面是点还是其他都是可以的，例如 oocss 里面就有经典的 content:".",有些版本可能 content 里面内容为空,一丝冰凉是不推荐这样做的,firefox 直到 7.0 content:”" 仍然会产生额外的空隙;

5）zoom：1 触发 IE hasLayout。

通过分析发现，除了 clear：both 用来闭合浮动的，其他代码无非都是为了隐藏掉 content 生成的内容，这也就是其他版本的闭合浮动为什么会有 font-size：0，line-height：0。

28.什么是外边距合并?

外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。

合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。

w3school 介绍网址：http://www.w3school.com.cn/css/css_margin_collapsing.asp

29.zoom:1 的清除浮动原理?

清除浮动，触发 hasLayout；

Zoom 属性是 IE 浏览器的专有属性，它可以设置或检索对象的缩放比例。解决 ie 下比较奇葩的 bug。

譬如外边距（margin）的重叠，浮动清除，触发 ie 的 haslayout 属性等。

来龙去脉大概如下：

当设置了 zoom 的值之后，所设置的元素就会就会扩大或者缩小，高度宽度就会重新计算了，这里一旦改变 zoom 值时其实也会发生重新渲染，运用这个原理，也就解决了 ie 下子元素浮动时候父元素不随着自动扩大的问题。

Zoom 属是 IE 浏览器的专有属性，火狐和老版本的 webkit 核心的浏览器都不支持这个属性。然而，zoom 现在已经被逐步标准化，出现在 CSS 3.0 规范草案中。

目前非 ie 由于不支持这个属性，它们又是通过什么属性来实现元素的缩放呢？

可以通过 css3 里面的动画属性 scale 进行缩放。

30.CSS 优化、提高性能的方法有哪些?

关键选择器（key selector）。选择器的最后面的部分为关键选择器（即用来匹配目标元素的部分）；

如果规则拥有 ID 选择器作为其关键选择器，则不要为规则增加标签。过滤掉无关的规则（这样样式系统就不会浪费时间去匹配它们了）；

提取项目的通用公有样式，增强可复用性，按模块编写组件；增强项目的协

同开发性、可维护性和可扩展性;

使用预处理工具或构建工具（gulp 对 css 进行语法检查、自动补前缀、打包压缩、自动优雅降级）；

31.怎么让 Chrome 支持小于 12px 的文字?

1、用图片：如果是内容固定不变情况下，使用将小于 12px 文字内容切出做图片，这样不影响兼容也不影响美观。

2、使用 12px 及 12px 以上字体大小：为了兼容各大主流浏览器，建议设计美工图时候设置大于或等于 12px 的字体大小，如果是接单的这个时候就需要给客户讲解小于 12px 浏览器不兼容等事宜。

3、继续使用小于 12px 字体大小样式设置：如果不考虑 chrome 可以不用考虑兼容，同时在设置小于 12px 对象设置-webkit-text-size-adjust:none，做到最大兼容考虑。

4、使用 12px 以上字体：为了兼容、为了代码更简单 从新考虑权重下兼容性。

32.position:fixed;在 android 下无效怎么处理?

fixed 的元素是相对整个页面固定位置的，你在屏幕上滑动只是在移动这个所谓的 viewport，

原来的网页还好好的在那，fixed 的内容也没有变过位置，

所以说并不是 iOS 不支持 fixed，只是 fixed 的元素不是相对手机屏幕固定的。

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no"/>

33.如果需要手动写动画，你认为最小时间间隔是多久，为什么？（阿里）

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

* 1000ms = 16.7ms

34.display:inline-block 什么时候会显示间隙？(携程)

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

35.有哪项方式可以对一个 DOM 设置它的 CSS 样式?

外部样式表，引入一个外部 css 文件

内部样式表，将 css 代码放在 <head> 标签内部

内联样式，将 css 样式直接定义在 HTML 元素内部

36.超链接访问过后 hover 样式就不出现的问题是什么？如何解决?

被点击访问过的超链接样式不再具有 hover 和 active 了,解决方法是改变 CSS 属性的排列顺序: L-V-H-A（link,visited,hover,active）

37.什么是 Css Hack？ie6,7,8 的 hack 分别是什么?

针对不同的浏览器写不同的 CSS code 的过程，就是 CSS hack。

示例如下:

```
#test{
    background-color:yellow;      /*ie8*/
    +background-color:pink;       /*ie7*/
    _background-color:orange;     /*ie6*/    }
```

更好的方式是使用 IE 条件判断语句:

<!—[if lte IE 6]>

内容

<![endif]–>

等

38.什么是外边距重叠？重叠的结果是什么?

外边距重叠就是 margin–collapse。

在 CSS 当中，相邻的两个盒子（可能是兄弟关系也可能是祖先关系）的外边距可以结合成一个单独的外边距。这种合并外边距的方式被称为折叠，并且因而所结合成的外边距称为折叠外边距。

折叠结果遵循下列计算规则：

两个相邻的外边距都是正数时，折叠结果是它们两者之间较大的值。

两个相邻的外边距都是负数时，折叠结果是两者绝对值的较大值。

两个外边距一正一负时，折叠结果是两者的相加的和。

39.rgba()和 opacity 的透明效果有什么不同?

rgba()和 opacity 都能实现透明效果，但最大的不同是 opacity 作用于元素，以及元素内的所有内容的透明度，

而 rgba()只作用于元素的颜色或其背景色。（设置 rgba 透明的元素的子元素不会继承透明效果！）

40.如何垂直居中一个浮动元素?

```
    // 方法一：已知元素的高宽
#div1{
    background–color:#6699FF;
```

```
    width:200px;
    height:200px;
    position: absolute;          /*父元素需要相对定位*/
    top: 50%;
    left: 50%;
    margin-top:-100px ;    /*二分之一的 height，width*/
    margin-left: -100px;
    }

//方法二:未知元素的高宽

  #div1{
    width: 200px;
    height: 200px;
    background-color: #6699FF;

    margin:auto;
    position: absolute;          /*父元素需要相对定位*/
    left: 0;
    top: 0;
```

```
        right: 0;
        bottom: 0;
        }
    /*如何垂直居中一个<img>?（用更简便的方法。）*/

    #container        /*<img>的容器设置如下*/
    {
        display:table-cell;
        text-align:center;
        vertical-align:middle;
    }
```

41.px 和 em 的区别

px 和 em 都是长度单位，区别是：

px 值固定，容易计算。

em 值不固定，是相对单位，其相对应父级元素的字体大小会调整

42.描述一个”reset”的 CSS 文件并如何使用它。知道 normalize.css 吗？你了解他们的不同之处?

Reset 样式的目的是清除浏览器某些默认样式，方便 css 书写，例如：

```
* {margin:0;padding:0;list-style:none;}
```

Normalize 的理念与 reset 不同，它并不清除浏览器默认样式，而是尽量将所

有浏览器的默认样式统一。

43.Sass、LESS 是什么？大家为什么要使用他们?

他们是 CSS 预处理器。他是 CSS 上的一种抽象层。他们是一种特殊的语法/语言编译成 CSS。

例如 Less 是一种动态样式语言. 将 CSS 赋予了动态语言的特性，如变量，继承,运算， 函数. LESS 既可以在客户端上运行（支持 IE 6+, Webkit, Firefox），也可一在服务端运行（借助 Node.js）。

为什么要使用它们？

结构清晰，便于扩展。

可以方便地屏蔽浏览器私有语法差异。这个不用多说，封装对浏览器语法差异的重复处理，减少无意义的机械劳动。

可以轻松实现多重继承。

完全兼容 CSS 代码，可以方便地应用到老项目中。LESS 只是在 CSS 语法上做了扩展，所以老的 CSS 代码也可以与 LESS 代码一同编译。

44.为什么要初始化样式?

用于浏览器默认 css 样式的存在并且不同浏览器对相同 HTML 标签的默认样式不同，若不初始化会造成不同浏览器之间的显示差异。

45.IE 的双边距 BUG：块级元素 float 后设置横向 margin，ie6 显示的 margin 比设置的较大。

解决：加入_display：inline

46.html 常见兼容性问题?

1.双边距 BUG float 引起的 使用 display

2.3 像素问题 使用 float 引起的 使用 dislpay:inline –3px

3.超链接 hover 点击后失效 使用正确的书写顺序 link visited hover active

4.IE z-index 问题 给父级添加 position:relative

5.Png 透明 使用 js 代码 改

6.Min-height 最小高度 ！ Important 解决’

7.select 在 ie6 下遮盖 使用 iframe 嵌套

8.为什么没有办法定义 1px 左右的宽度容器（IE6 默认的行高造成的，使用 over:hidden,zoom:0.08 line-height:1px）

9.IE5-8 不支持 opacity，解决办法：

```
.opacity {
    opacity: 0.4
    filter: alpha(opacity=60); /* for IE5-7 */
    -ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; /* for IE 8*/
}
```

10. IE6 不支持 PNG 透明背景，解决办法: IE6 下使用 gif 图片

47.对 WEB 标准以及 W3C 的理解与认识

标签闭合、标签小写、不乱嵌套、提高搜索机器人搜索几率、使用外 链 css 和 js 脚本、结构行为表现的分离、文件下载与页面速度更快、内容能被更多的用户所访问、内容能被更广泛的设备所访问、更少的代码和组件，容易维 护、改版方便，不需要变动页面内容、提供打印版本而不需要复制内容、提高网站易用性。

48.行内元素和块级元素的具体区别是什么？行内元素的 padding 和 margin 可设置吗?

块级元素(block)特性：

总是独占一行，表现为另起一行开始，而且其后的元素也必须另起一行显示;

宽度(width)、高度(height)、内边距(padding)和外边距(margin)都可控制;

内联元素(inline)特性：

和相邻的内联元素在同一行;

宽度(width)、高度(height)、内边距的top/bottom(padding-top/padding-bottom)和外边距的top/bottom(margin-top/margin-bottom)都不可改变（也就是 padding 和 margin 的 left 和 right 是可以设置的），就是里面文字或图片的大小。

那么问题来了，浏览器还有默认的天生 inline-block 元素（拥有内在尺寸，可设置高宽，但不会自动换行），有哪些？

答案：<input> 、<img> 、<button> 、<textarea> 、<label>

49.什么是外边距重叠？重叠的结果是什么?

答案：

外边距重叠就是 margin-collapse。

在 CSS 当中，相邻的两个盒子（可能是兄弟关系也可能是祖先关系）的外边距可以结合成一个单独的外边距。这种合并外边距的方式被称为折叠，并且因而所结合成的外边距称为折叠外边距。

折叠结果遵循下列计算规则：

两个相邻的外边距都是正数时，折叠结果是它们两者之间较大的值。

两个相邻的外边距都是负数时，折叠结果是两者绝对值的较大值。

两个外边距一正一负时，折叠结果是两者的相加的和。

50.知道 css 有个 content 属性吗？有什么作用？有什么应用？

知道。css 的 content 属性专门应用在 before/after 伪元素上，用来插入生成内容。最常见的应用是利用伪类清除浮动。

```
//一种常见利用伪类清除浮动的代码

.clearfix:after {

    content:"."; //这里利用到了 content 属性

    display:block;

    height:0;

    visibility:hidden;

    clear:both; }

.clearfix {

    *zoom:1;

}
```

after 伪元素通过 content 在元素的后面生成了内容为一个点的块级素，再利用 clear:both 清除浮动。

那么问题继续还有，知道 css 计数器（序列数字字符自动递增）吗？如何通过 css content 属性实现 css 计数器？

答案：css 计数器是通过设置 counter-reset 、counter-increment 两个属性 、及 counter()/counters()一个方法配合 after / before 伪类实现。

三 JavaScript

1.js 基础

1.eval 是做什么的?

它的功能是把对应的字符串解析成 JS 代码并运行;

应该避免使用 eval，不安全，非常耗性能（2 次，一次解析成 js 语句，一次执行）。

由 JSON 字符串转换为 JSON 对象的时候可以用 eval，var obj =eval('('+ str +')');

2.什么是 window 对象？什么是 document 对象?

window 对象是指浏览器打开的窗口。

document 对象是 Documentd 对象（HTML 文档对象）的一个只读引用，window 对象的一个属性。

3.null，undefined 的区别?

null 表示一个对象是“没有值”的值，也就是值为“空”;

undefined 表示一个变量声明了没有初始化(赋值);

undefined 不是一个有效的 JSON，而 null 是;

undefined 的类型(typeof)是 undefined;

null 的类型(typeof)是 object;

Javascript 将未赋值的变量默认值设为 undefined;

Javascript 从来不会将变量设为 null。它是用来让程序员表明某个用 var 声明的变量时没有值的。

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

null == undefined // true

null === undefined // false

再来一个例子:

null

Q：有张三这个人么？

A：有!

Q：张三有房子么？

A：没有!

undefined

Q：有张三这个人么？

A：有！

Q: 张三有多少岁？

A: 不知道（没有被告诉）

4.["1", "2", "3"].map(parseInt) 答案是多少?

parseInt() 函数能解析一个字符串，并返回一个整数，需要两个参数（val, radix），

其中 radix 表示要解析的数字的基数。【该值介于 2 ~ 36 之间，并且字符串中的数字不能大于 radix 才能正确返回数字结果值】；

但此处 map 传了 3 个 (element, index, array),我们重写 parseInt 函数测试一下是否符合上面的规则。

```
function parseInt(str, radix) {
    return str+'-'+radix;
};
var a=["1", "2", "3"];
a.map(parseInt);  // ["1-0", "2-1", "3-2"] 不能大于 radix
```

因为二进制里面，没有数字 3,导致出现超范围的 radix 赋值和不合法的进制解析，才会返回 NaN

所以["1", "2", "3"].map(parseInt) 答案也就是：[1, NaN, NaN]

5.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：I.E.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;）

6.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。

7.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。

8.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);

9.js 延迟加载的方式有哪些？

defer 和 async、动态创建 DOM 方式（用得最多）、按需异步载入 js

10.如何判断当前脚本运行在浏览器还是 node 环境中？（阿里）

this === window ? 'browser' : 'node';

通过判断 Global 对象是否为 window，如果不为 window，当前脚本没有运行在浏览器中

11.javascript 的 typeof 返回哪些数据类型

alert(typeof [1, 2]); //object

alert(typeof 'leipeng'); //string

var i = true;

alert(typeof i); //boolean

alert(typeof 1); //number

var a;

alert(typeof a); //undefined

function a(){;};

alert(typeof a) //function

12.例举 3 种强制类型转换和 2 种隐式类型转换?

强制（parseInt(),parseFloat(),Number()）

隐式（== ,!!）

13.split() 、join() 的区别

前者是切割成数组的形式，后者是将数组转换成字符串

14.数组方法 pop() push() unshift() shift()

push()尾部添加 pop()尾部删除

unshift()头部添加 shift()头部删除

map()：遍历数组中的元素，返回一个新数组(包含回调函数返回的数据)

filter():遍历数组中的元素，返回一个新数组(包含回调函数返回 true 的元素)

15.事件绑定和普通事件有什么区别

普通添加事件的方法：

```
var btn = document.getElementById("hello");
btn.onclick = function(){
    alert(1);
}
btn.onclick = function(){
    alert(2);
```

}

执行上面的代码只会 alert 2

事件绑定方式添加事件：

```
var btn = document.getElementById("hello");
btn.addEventListener("click",function(){
    alert(1);
},false);
btn.addEventListener("click",function(){
    alert(2);
},false);
```

执行上面的代码会先 alert 1 再 alert 2

普通添加事件的方法不支持添加多个事件，最下面的事件会覆盖上面的，而事件绑定（addEventListener）方式添加事件可以添加多个。

addEventListener 不兼容低版本 IE

普通事件无法取消

addEventLisntener 还支持事件冒泡+事件捕获

16.IE 和 DOM 事件流的区别

1.执行顺序不一样、

2.参数不一样

3.事件加不加 on

4.this 指向问题

17.IE 和标准下有哪些兼容性的写法

var ev = ev || window.event

document.documentElement.clientWidth || document.body.clientWidth

var target = ev.srcElement||ev.target

18.如何阻止事件冒泡和事件默认行为

```
//阻止事件冒泡
if(typeof ev.stopPropagation=='function') {   //标准的
    ev.stopPropagation();
} else { //非标准 IE
    window.event.cancelBubble = true;
}
//阻止事件默认行为
return false
```

19.window.onload 和 document ready 的区别

window.onload 是在 dom 文档树加载完和所有文件加载完之后执行一个函数 document.ready 原生中没有这个方法，jquery 中有 $().ready(function)，在 dom 文档树加载完之后执行一个函数（注意，这里面的文档树加载完不代表全部文件加载完）。

$(document).ready 要比 window.onload 先执行

window.onload 只能出来一次，$(document).ready 可以出现多次

20.”==”和“===”的不同

前者会自动转换类型

后者不会

21.JavaScript 是一门什么样的语言，它有哪些特点？

javaScript 一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为 JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在 HTML 网页上使用，用来给 HTML 网页增加动态功能。JavaScript 兼容于 ECMA 标准，因此也称为 ECMAScript。

基本特点

1. 是一种解释性脚本语言（代码不进行预编译）。

2. 主要用来向 HTML（标准通用标记语言下的一个应用）页面添加交互行为。

3. 可以直接嵌入 HTML 页面，但写成单独的 js 文件有利于结构和行为的分离。

跨平台特性，在绝大多数浏览器的支持下，可以在多种平台下运行（如 Windows、Linux、Mac、Android、iOS 等）。

22.JavaScript 的数据类型都有什么？

基本数据类型：String,boolean,Number,Undefined, Null

引用数据类型：Object, Array, Function

那么问题来了，如何判断某变量是否为数组数据类型？

方法一.判断其是否具有“数组性质”，如 slice()方法。可自己给该变量定义 slice 方法，故有时会失效

方法二.obj instanceof Array 在某些 IE 版本中不正确

方法三.方法一二皆有漏洞，在 ECMA Script5 中定义了新方法 Array.isArray()，保证其兼容性，最好的方法如下：

```
if(typeof Array.isArray==="undefined")
{
  Array.isArray = function(arg){
      return Object.prototype.toString.call(arg)==="[object Array]"
  };
}
```

23.当一个 DOM 节点被点击时候，我们希望能够执行一个函数，应该怎么做？

直接在 DOM 里绑定事件：

在 JS 里通过 onclick 绑定：xxx.onclick = test

通过事件添加进行绑定：addEventListener(xxx, ‘click’, test)

那么问题来了，Javascript 的事件流模型都有什么？

“事件冒泡”：事件开始由最具体的元素接受，然后逐级向上传播

“事件捕捉”：事件由最不具体的节点先接收，然后逐级向下，一直到最具体的

“DOM 事件流”：三个阶段：事件捕捉，目标阶段，事件冒泡

24.看下列代码输出为何？解释原因。

```
var a;
alert(typeof a); // undefined
alert(b); // 报错
```

解释：Undefined 是一个只有一个值的数据类型，这个值就是“undefined”，在使用 var 声明变量但并未对其赋值进行初始化时，这个变量的值就是 undefined。而 b 由于未声明将报错。注意未申明的变量和声明了未赋值的是不一样的。

25.看下列代码,输出什么？解释原因。

```
var undefined;

undefined == null; // true

1 == true;     // true

2 == true;     // false

0 == false;    // true

0 == '';       // true

NaN == NaN;    // false

[] == false; // true

[] == ![];     // true
```

undefined 与 null 相等，但不恒等（===）

一个是 number 一个是 string 时，会尝试将 string 转换为 number

尝试将 boolean 转换为 number，0 或 1

尝试将 Object 转换成 number 或 string，取决于另外一个对比量的类型

所以，对于 0、空字符串的判断，建议使用 “===” 。“===”会先判断两边的值类型，类型不匹配时为 false。

那么问题来了，看下面的代码，输出什么，foo 的值为什么？

```
var foo = "11"+2-"1";

console.log(foo);

console.log(typeof foo);
```

执行完后 foo 的值为 111，foo 的类型为 String。

26.已知数组 var stringArray = [“This”，“is”，“Baidu”，“Campus”]，Alert 出”This is Baidu Campus”。

```
alert(stringArray.join(“ ”))
```

27.已知有字符串 foo=”get-element-by-id”,写一个 function 将其转化成驼峰表示法”getElementById”

```
function combo(msg){

    var arr=msg.split("-");

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

      arr[i]=arr[i].charAt(0).toUpperCase()+arr[i]

.substr(1,arr[i].length-1);

    }

    msg=arr.join("");

    return msg;

}
```

28.输出今天的日期，以 YYYY-MM-DD 的方式，比如今天是 2014 年 9 月 26 日，则输出 2014-09-26

```
var d = new Date();

// 获取年，getFullYear()返回 4 位的数字
```

```
var year = d.getFullYear();

// 获取月，月份比较特殊，0 是 1 月，11 是 12 月

var month = d.getMonth() + 1;

var d = new Date();

// 获取年，getFullYear()返回 4 位的数字

var year = d.getFullYear();

// 获取月，月份比较特殊，0 是 1 月，11 是 12 月

var month = d.getMonth() + 1;
```

29.将字符串”<tr><td>{$id}</td><td>{$name}</td></tr>”中的{$id}替换成 10，{$name}替换成 Tony （使用正则表达式）

```
"<tr><td>{$id}</td><td>{$id}_{$name}</td></tr>".replace(/{\$id}/g,
'10').replace(/{\$name}/g, 'Tony');
```

30.为了保证页面输出安全，我们经常需要对一些特殊的字符进行转义，请写一个函数 escapeHtml，将<, >, &, “进行转义

```
function escapeHtml(str) {

return str.replace(/[<>”&]/g, function(match) {

    switch (match) {

                case “<”:

                    return “&lt;”;

                case “>”:

                    return “&gt;”;

                case “&”:
```

```
                return “&amp;”;
            case “\””:
                return “&quot;”;
        }
    });
}
```

31.foo = foo||bar ，这行代码是什么意思？为什么要这样写？

如果 foo 存在，值不变，否则把 bar 的值赋给 foo。

短路表达式：作为”&&”和”||”操作符的操作数表达式，这些表达式在进行求值时，只要最终的结果已经可以确定是真或假，求值过程便告终止，这称之为短路求值。

32.看下列代码，将会输出什么?(变量声明提升)

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

答案：输出 undefined 和 2。上面代码相当于：

```
var foo = 1;
(function(){
```

```
    var foo;
    console.log(foo); //undefined
    foo = 2;
    console.log(foo); // 2;
})()
```

函数声明与变量声明会被 JavaScript 引擎隐式地提升到当前作用域的顶部，但是只提升名称不会提升赋值部分。

33.用 js 实现随机选取 10–100 之间的 10 个数字，存入一个数组，并排序。

```
function randomNub(aArray, len, min, max) {
        if (len >= (max – min)) {
            return '超过' + min + '–' + max + '之间的个数范围' + (max – min – 1) + '个的总数';
        }
        if (aArray.length >= len) {
            aArray.sort(function(a, b) {
                return a – b
            });
            return aArray;
        }
        var nowNub = parseInt(Math.random() * (max – min – 1)) + (min + 1);
        for (var j = 0; j < aArray.length; j++) {
```

```
            if (nowNub == aArray[j]) {
                randomNub(aArray, len, min, max);
                return;
            }
        }
        aArray.push(nowNub);
        randomNub(aArray, len, min, max);
        return aArray;

        var arr=[];
        randomNub(arr,10,10,100);
```

34.把两个数组合并，并删除第二个元素。

```
var array1 = ['a','b','c'];
var bArray = ['d','e','f'];
var cArray = array1.concat(bArray);
cArray.splice(1,1);
```

35. 有这样一个URL：http://item.taobao.com/item.htm?a=1&b=2&c=&d=xxx&e，请写一段JS程序提取URL中的各个GET参数(参数名和参数个数不确定)，将其按key-value形式返回到一个json结构中，如{a:’1’，b:’2’，c:”，d:’xxx’，e:undefined}。

```
function serilizeUrl(url) {
    var urlObject = {};
```

```
    if (/\?/.test(url)) {
        var urlString = url.substring(url.indexOf("?") + 1);
        var urlArray = urlString.split("&");
        for (var i = 0, len = urlArray.length; i < len; i++) {
            var urlItem = urlArray[i];
            var item = urlItem.split("=");
            urlObject[item[0]] = item[1];
        }
        return urlObject;
    }
    return null;
}
```

36.正则表达式构造函数 var reg=new RegExp(“xxx”)与正则表达字面量 var reg=//有什么不同？匹配邮箱的正则表达式？

当使用 RegExp()构造函数的时候，不仅需要转义引号（即\”表示”），并且还需要双反斜杠（即\\表示一个\）。使用正则表达字面量的效率更高。

邮箱的正则匹配：

```
var regMail = /^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+((.[a-zA-Z0-9_-]{2,3}){1,2})$/;
```

37.写一个 function，清除字符串前后的空格。

```
if (!String.prototype.trim) {
    String.prototype.trim = function() {
```

```
        return this.replace(/^\s+/, "").replace(/\s+$/,"");
    }
}
//测试
var str = " \t\n test string ".trim();
alert(str == "test string"); // alerts "true"
```

38.以下两个变量 a 和 b，a+b 的哪个结果是 NaN？

A、var a=undefined; b=NaN

B、var a= ‘123’; b=NaN

C、var a =undefined , b =NaN

D、var a=NaN , b='undefined'

39.下面的 JavaScript 语句中，（）实现检索当前页面中的表单元素中的所有文本框，并将它们全部清空

```
A. for(vari=0;i< form1.elements.length;i++) {
if(form1.elements.type==”text”)
form1.elements.value=”";}
B. for(vari=0;i<document.forms.length;i++) {
if(forms[0].elements.type==”text”)
forms[0].elements.value=”";
}
C. if(document.form.elements.type==”text”)
```

form.elements.value=”";

D. for(vari=0;i<document.forms.length; i++){

for(var j=0;j<document.forms.elements.length; j++){

if(document.forms.elements[j].type==”text”)

document.forms.elements[j].value=”";

}

}

40.typeof 运算符返回值中有一个跟 javascript 数据类型不一致，它是

Array

41.写出函数 DateDemo 的返回结果，系统时间假定为今天

function DateDemo(){

var d, s="今天日期是：";

d = new Date();

s += d.getMonth() +1+ "/";

s += d.getDate() + "/";

s += d.getFullYear();

return s;

}

结果：今天日期是：7/21/2016

42.写出简单描述 html 标签（不带属性的开始标签和结束标签）的正则表达式，并将以下字符串中的 html 标签去除掉

var str = “ 这里是 div 里面的段落 ”;

<scripttype=”text/javascript”>

var reg = /<\/?\w+\/?>/gi;

var str = “ 这里是 div 里面的段落 ”;

alert(str.replace(reg,”"));

</script>

43.截取字符串 abcdefg 的 efg

alert('abcdefg'.substring(4));

44.简述创建函数的几种方式

第一种（函数声明）：

function sum1(num1,num2){

return num1+num2;

}

第二种（函数表达式）：

var sum2 = function(num1,num2){

return num1+num2;

}

第三种（函数对象方式）：

var sum3 = new Function("num1","num2","return num1+num2");

45.Javascript 如何实现继承？

1.构造继承法

2.原型继承法

3.实例继承法

46.Javascript 创建对象的几种方式？

1、var obj = {};（使用 json 创建对象）

如：obj.name = '张三';

obj.action = function ()

{

alert('吃饭');

} ;

2、var obj = new Object();（使用 Object 创建对象）

如：obj.name = '张三';

obj.action = function ()

{

alert('吃饭');

} ;

3、 通过构造函数创建对象。

(1)、使用 this 关键字

如：var obj = function (){

this.name ='张三';

this.age = 19;

this.action = function ()

{

alert('吃饭');

} ;

}

(2)、使用 prototype 关键字

如：function obj (){}

obj.prototype.name ='张三';

obj.prototype.action=function ()

{

alert('吃饭');

} ;

4、使用内置对象创建对象。

如：var str = new String("实例初始化 String");

var str1 = "直接赋值的 String";

var func = new Function("x","alert(x)");//示例初始化 func

var obj = new Object();//示例初始化一个 Object

47.js 延迟加载的方式有哪些?

1. defer 和 async

2. 动态创建 DOM 方式(创建 script,插入到 DOM 中,加载完毕后 callBack)

3. 按需异步载入 js

48.哪些操作会造成内存泄漏?

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

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。

1. 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。

2. 闭包

3. 控制台日志

4. 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）

49.判断一个字符串中出现次数最多的字符，统计这个次数

```
var str = 'asdfssaaasasasasaa';

var json = {};

for (var i = 0; i < str.length; i++) {

        if(!json[str.charAt(i)]){

                json[str.charAt(i)] = 1;

        }else{

                json[str.charAt(i)]++;
```

```
            }
    };
    var iMax = 0;
    var iIndex = '';
    for(var i in json){
            if(json[i]>iMax){
                    iMax = json[i];
                    iIndex = i;
            }
    }
    alert('出现次数最多的是:'+iIndex+'出现'+iMax+'次');
```

50.写一个获取非行间样式的函数

```
    function getStyle(obj,attr,value)
    {
        if(!value)
        {
            if(obj.currentStyle)
            {
                return obj.currentStyle(attr);
            }
```

```
            else{
                obj.getComputedStyle(attr,false);
            }
        }
        else
        {
            obj.style[attr] = value;
        }
    }
```

51.字符串反转，如将 '12345678' 变成 '87654321'

```
//思路：先将字符串转换为数组 split()，利用数组的反序函数 reverse()颠倒数组，再利用 jion() 转换为字符串
var str = '12345678';
str = str.split('').reverse().join('');
```

52.将数字 12345678 转化成 RMB 形式 如： 12,345,678

```
//思路：先将数字转为字符， str= str + '' ;
//利用反转函数，每三位字符加一个 ','最后一位不加； re()是自定义的反转函数，最后再反转回去!
function re(str) {
    str += '';
    return str.split("").reverse().join("");
```

```
}


function toRMB(num) {
    var tmp='';
    for (var i = 1; i <= re(num).length; i++) {
        tmp += re(num)[i - 1];
        if (i % 3 == 0 && i != re(num).length) {
            tmp += ',';
        }
    }
    return re(tmp);
}
```

53.生成 5 个不同的随机数；

```
//思路：5 个不同的数，每生成一次就和前面的所有数字相比较，如果有相同的，则放弃当前生成的数字！
var num1 = [];
for(var i = 0; i < 5; i++){
    num1[i] = Math.floor(Math.random()*10) + 1; //范围是 [1, 10]
    for(var j = 0; j < i; j++){
        if(num1[i] == num1[j]){
            i--;
```

```
            }
        }
    }
```

54.去掉数组中重复的数字

方法一:

```
//思路：每遍历一次就和之前的所有做比较，不相等则放入新的数组中!
//这里用的原型 个人做法;
Array.prototype.unique = function(){
    var len = this.length,
        newArr = [],
        flag = 1;
    for(var i = 0; i < len; i++, flag = 1){
        for(var j = 0; j < i; j++){
            if(this[i] == this[j]){
                flag = 0;           //找到相同的数字后，不执行添加数据
            }
        }
        flag ? newArr.push(this[i]) : '';
    }
    return newArr;
```

```
}
```

方法二：

```
var arr=[1,2,3,3,4,4,5,5,6,1,9,3,25,4];
Array.prototype.unique2 = function()
{
    var n = []; //一个新的临时数组
    for(var i = 0; i < this.length; i++) //遍历当前数组
    {
        //如果当前数组的第 i 已经保存进了临时数组，那么跳过，
        //否则把当前项 push 到临时数组里面
        if (n.indexOf(this[i]) == -1) n.push(this[i]);
    }
    return n;
}
var newArr2=arr.unique2(arr);
alert(newArr2); //输出 1,2,3,4,5,6,9,25
```

55.阶乘函数；

```
//原型方法
Number.prototype.N = function(){
    var re = 1;
```

```
        for(var i = 1; i <= this; i++){
            re *= i;
        }
        return re;
    }
    var num = 5;
    alert(num.N());
```

56.window.location.search() 返回的是什么?

http://localhost:8080/xxx?ver=1.0&id=123

返回值：?ver=1.0&id=timlq 也就是问号后面的部分

57.window.location.reload() 作用?

新当前页面。

58.javascript 中的垃圾回收机制?

在 Javascript 中，如果一个对象不再被引用，那么这个对象就会被 GC 回收。如果两个对象互相引用，而不再 被第 3 者所引用，那么这两个互相引用的对象也会被回收。因为函数 a 被 b 引用，b 又被 a 外的 c 引用，这就是为什么 函数 a 执行后不会被回收的原因。

59.精度问题: JS 精度不能精确到 0.1 所以 。。。。同时存在于值和差值中

```
var n = 0.3,m = 0.2, i = 0.2, j = 0.1;
alert((n - m) == (i - j)); //false
alert((n-m) == 0.1); //false
alert((i-j)==0.1); //true
```

60.计算字符串字节数：

```
new function(s){
    if(!arguments.length||!s) return null;
    if(""==s) return 0;
    var l=0;
    for(var i=0;i<s.length;i++){
        if(s.charCodeAt(i)>255) l+=2; else l+=1;    //charCodeAt()得到的是 unCode 码
    }       //汉字的 unCode 码大于 255bit 就是两个字节
    alert(l);
}("hello world!");
```

61.匹配输入的字符：第一个必须是字母或下划线开头，长度 5-20

```
var reg = /^[a-zA-Z_][a-zA-Z0-9_]{5,20}/,
        name1 = 'leipeng',
        name2 = '0leipeng',
        name3 = '你好 leipeng',
        name4 = 'hi';

    alert(reg.test(name1));
    alert(reg.test(name2));
    alert(reg.test(name3));
```

```
alert(reg.test(name4));
```

62.如何在 HTML 中添加事件，几种方法？

1、标签之中直接添加 onclick="fun()";

2、JS 添加 Eobj.onclick = method;

3、绑定事件 IE： obj.attachEvent('onclick', method);

FF: obj.addEventListener('click', method, false);

63.你如何优化自己的代码？

代码重用

避免全局变量（命名空间，封闭空间，模块化 mvc..）

拆分函数避免函数过于臃肿

注释

64.使用 js 实现这样的效果：在文本域里输入文字时，当按下 enter 键时不换行，而是替换成“{{enter}}”,(只需要考虑在行尾按下 enter 键的情况).

```
<html>
<head>
    <script>
        function back(ele,event){
            event = event || window.event;
            if(event.keyCode==13){
                event.returnValue = false;
                ele.value+="{{enter}}"
```

```
                return false;
            }
        }
    </script>
</head>
<body>
<textarea rows="3" cols="40" id="te" onkeypress="back(this,event);"></textarea>
</body>
</html>
```

65.简述 readyonly 与 disabled 的区别

ReadOnly 和 Disabled 的作用是使用户不能够更改表单域中的内容.

但是二者还是有着一些区别的:

1、Readonly 只针对 input(text/password)和 textarea 有效，而 disabled 对于所有的表单元素有效，包括 select,radio,checkbox,button 等。

2、在表单元素使用了 disabled 后，我们将表单以 POST 或者 GET 的方式提交的话，这个元素的值不会被传递出去，而 readonly 会将该值传递出去

66.为什么扩展 javascript 内置对象不是好的做法?

因为你不知道哪一天浏览器或 javascript 本身就会实现这个方法，而且和你扩展的实现有不一致的表现。到时候你的 javascript 代码可能已经在无数个页面中执行了数年，而浏览器的实现导致所有使用扩展原型的代码都崩溃了。。。

67.什么是三元表达式？“三元”表示什么意思？

三元运算符:

三元如名字表示的三元运算符需要三个操作数。

语法是 条件 ？ 结果 1 : 结果 2;. 这里你把条件写在问号(?)的前面后面跟着用冒号(:)分隔的结果 1 和结果 2。满足条件时结果 1 否则结果 2。

68.变量的命名规范以及命名推荐

变量，函数，方法：小写开头，以后的每个单词首字母大写 （驼峰）

构造函数，class：每个单词大写开头

基于实际情况，以动词，名词，谓词来命名。尽量言简意骇，以命名代替注释

69.三种弹窗的单词以及三种弹窗的功能

1.alert

//弹出对话框并输出一段提示信息

```
function ale() {
    //弹出一个对话框
    alert("提示信息！");

}
```

2.confirm

//弹出一个询问框，有确定和取消按钮

```
function firm() {
    //利用对话框返回的值 （true 或者 false）
    if (confirm("你确定提交吗？")) {
        alert("点击了确定");
    }
    else {
        alert("点击了取消");
    }

}
```

3.prompt

```
    //弹出一个输入框，输入一段文字，可以提交
    function prom() {
        var name = prompt("请输入您的名字", ""); //将输入的内容赋给变量 name ，

        //这里需要注意的是，prompt 有两个参数，前面是提示的话，后面是当对话框出来后，在对话框里的默认值
        if (name)//如果返回的有内容
        {
```

```
                alert("欢迎您：" + name)
            }
        }
```

70.主流浏览器内核

IE trident　　火狐 gecko　　谷歌苹果 webkit　　Opera：Presto

71.JavaScript 的循环语句有哪些？

for,for..in,while,do...while

72.闭包：下面这个 ul，如何点击每一列的时候 alert 其 index？

```
<ul id="test">
    <li>这是第一条</li>
    <li>这是第二条</li>
    <li>这是第三条</li>
</ul>
//js
 window.onload = function() {
        var lis = document.getElementById('test').children;
        for (var i = 0; i < lis.length; i++) {
            lis[i].onclick = (function(i) {
                return function() {
                    alert(i)
```

```
                    };
                })(i);
            };
        }
```

73.列出 3 条以上 ff 和 IE 的脚本兼容问题

(1) window.event：

表示当前的事件对象，IE 有这个对象，FF 没有，FF 通过给事件处理函数传递事件对象

(2) 获取事件源

IE 用 srcElement 获取事件源，而 FF 用 target 获取事件源

(3) 添加，去除事件

IE：element.attachEvent(“onclick”, function) element.detachEvent(“onclick”, function)

FF：element.addEventListener(“click”, function, true) element.removeEventListener(“click”, function, true)

(4) 获取标签的自定义属性

IE：div1.value 或 div1[“value”]

FF：可用 div1.getAttribute(“value”)

74.在 Javascript 中什么是伪数组？如何将伪数组转化为标准数组？

伪数组（类数组）：无法直接调用数组方法或期望 length 属性有什么特殊的行为，但仍可以对真正数组遍历方法来遍历它们。典型的是函数的 argument 参数，还有像调用 getElementsByTagName,document.childNodes 之类的，它们都返回 NodeList 对象都属于伪数组。可以使用 Array.prototype.slice.call(fakeArray)将数组转化为真正的 Array 对象。

75.请写出一个程序，在页面加载完成后动态创建一个 form 表单，并在里面添加一个 input 对象并给它任意赋值后义 post 方式提交到：http://127.0.0.1/save.php

```
window.onload=function(){

    var form=document.createElement("form");

    form.setAttribute("method", "post");

    form.setAttribute("action", "http://127.0.0.1/save.php");



    var input=document.createElement("input");

    form.appendChild(input);

    document.body.appendChild(form);

    input.value="cxc";

    form.submit();//提交表单

}
```

76.用 JavaScript 实现冒泡排序。数据为 23、45、18、37、92、13、24

```
//升序算法

function sort(arr){
```

```
        for (var i = 0; i <arr.length; i++) {
            for (var j = 0; j <arr.length-i; j++) {
                if(arr[j]>arr[j+1]){
                    var c=arr[j];//交换两个变量的位置
                    arr[j]=arr[j+1];
                    arr[j+1]=c;
                }
            };
        };
        return arr.toString();
    }
    console.log(sort([23,45,18,37,92,13,24]));
```

77.前端代码优化的方法

```
    var User = {
        count = 1,
        getCount: function ( ) {
            return this.count;
        }
    }
    console.log(User.getCount());
```

```
var func = User.getCount;
console.log(func());
```

1 undefined（因为是 window 对象执行了 func 函数）；

78.用程序实现找到 html 中 id 名相同的元素？

```
<body>
    <form id='form1'>
        <div id='div1'></div>
        <div id='div2'></div>
        <div id='div3'></div>
        <div id='div4'></div>
        <div id='div5'></div>
        <div id='div3'>id 名重复的元素</div>
    </form>
</body>
var nodes=document.querySelectorAll("#form1>*");
for(var i=0,len=nodes.length;i<len;i++){
    var attr=nodes[i].getAttribute("id");
    var s=1;
    for(var j=i+1;j<len;j++){
        if(nodes[j].getAttribute("id")==attr){
```

```
                s++;
                alert("id 为："+attr+"的元素出现"+s+"次");
            }
        }
    }
```

79.程序中捕获异常的方法?

window.error

try{}catch(){}finally{}

80.下列控制台都输出什么

第 1 题：

```
function setName(){
    name="张三";
}
setName();
console.log(name);
```

答案："张三"

第 2 题：

```
//考点：1、变量声明提升 2、变量搜索机制
var a=1;
function test(){
```

```
    console.log(a);
    var a=1;
}
test();
```

答案：undefined

第 3 题：

```
var b=2;
function test2(){
    window.b=3;
    console.log(b);
}
test2();
```

答案：3

第 4 题：

```
c=5;//声明一个全局变量 c
function test3(){
    window.c=3;
    console.log(c);          //答案：undefined，原因：由于此时的 c 是一个局部变量 c，并且没有被赋值
    var c;
    console.log(window.c);//答案：3，原因：这里的 c 就是一个全局变量
```

```
c
}
test3();
```

第 5 题：

```
var arr = [];
arr[0]  = 'a';
arr[1]  = 'b';
arr[10] = 'c';
alert(arr.length); //答案：11
console.log(arr[5]); //答案：undefined
```

第 6 题：

```
var a=1;
console.log(a++);            //答案：1
console.log(++a);            //答案：3
```

第 7 题：

```
console.log(null==undefined); //答案：true
console.log("1"==1);          //答案：true，因为会将数字 1 先转换为字符串 1
console.log("1"===1);         //答案：false，因为数据类型不一致
```

第 8 题：

```
typeof 1;             "number"
```

typeof "hello";　　"string"

typeof /[0-9]/;　　"object"

typeof {};　　"object"

typeof null;　　"object"

typeof undefined;　　"undefined"

typeof [1,2,3];　　"object"

typeof function(){}; //"function"

第 9 题：

parseInt(3.14);　　//3

parseFloat("3asdf");　　//3

parseInt("1.23abc456");

parseInt(true);//"true" NaN

第 10 题：

//考点：函数声明提前

```
function bar() {
    return foo;
    foo = 10;
    function foo() {}
    //var foo = 11;
}
```

```
alert(typeof bar());//"function"
```

第 11 题：

```
//考点：函数声明提前
var foo = 1;
function bar() {
    foo = 10;
    return;
    function foo() {}
}
bar();
alert(foo);//答案：1
```

第 12 题：

```
console.log(a);//是一个函数
var a = 3;
function a(){}
console.log(a);////3
```

第 13 题：

```
//考点：对 arguments 的操作
function foo(a) {
    arguments[0] = 2;
```

alert(a);//答案：2，因为：a、arguments 是对实参的访问，b、通过 arguments[i]可以修改指定实参的值

}

foo(1);

第 14 题：

function foo(a) {

alert(arguments.length);//答案：3，因为 arguments 是对实参的访问

}

foo(1, 2, 3);

第 15 题

bar();//报错

var foo = function bar(name) {

console.log("hello"+name);

console.log(bar);

};

//alert(typeof bar);

foo("world");//"hello"

console.log(bar);//undefined

console.log(foo.toString());

bar();//报错

第 16 题：

```
function test(){
    console.log("test 函数");
}
setTimeout(function(){
    console.log("定时器回调函数");
}, 0)
test();
```

结果：

test 函数

定时器回调函数

81.请说出三种减低页面加载时间的方法

1、压缩 css、js 文件

2、合并 js、css 文件，减少 http 请求

3、外部 js、css 文件放在最底下

4、减少 dom 操作，尽可能用变量替代不必要的 dom 操作

82.实现一个函数 clone，可以对 JavaScript 中的 5 种主要的数据类型（包括 Number、String、Object、Array、Boolean）进行值复制

```
    // 方法一：
Object.prototype.clone = function(){
    var o = this.constructor === Array ? [] : {};
    for(var e in this){
```

```
        o[e] = typeof this[e] === "object" ? this[e].clone() : this[e];
    }
    return o;
}
//方法二:
        /**
        * 克隆一个对象
        * @param Obj
        * @returns
        */
      function clone(Obj) {
            var buf;
            if (Obj instanceof Array) {
                buf = [];//创建一个空的数组
                var i = Obj.length;
                while (i--) {
                    buf[i] = clone(Obj[i]);
                }
                return buf;
            }else if (Obj instanceof Object){
```

```
            buf = {};//创建一个空对象
            for (var k in Obj) { //为这个对象添加新的属性
                buf[k] = clone(Obj[k]);
            }
            return buf;
        }else{ //普通变量直接赋值
            return Obj;
        }
    }
```

83.如何消除一个数组里面重复的元素？

```
var arr=[1,2,3,3,4,4,5,5,6,1,9,3,25,4];
        function deRepeat(){
            var newArr=[];
            var obj={};
            var index=0;
            var l=arr.length;
            for(var i=0;i<l;i++){
                if(obj[arr[i]]==undefined)
                  {
                    obj[arr[i]]=1;
```

```
                newArr[index++]=arr[i];
            }
        else if(obj[arr[i]]==1)
            continue;
    }
    return newArr;
}
var newArr2=deRepeat(arr);
alert(newArr2); //输出 1,2,3,4,5,6,9,25
```

84.定义一个 log 方法，让它可以代理 console.log 的方法。

```
function log(msg)  {
    console.log(msg);
}
log("hello world!") // hello world!
```

如果要传入多个参数呢？显然上面的方法不能满足要求，所以更好的方法是：

```
function log(){
    console.log.apply(console, arguments);
};
```

到此，追问 apply 和 call 方法的异同。

对于 apply 和 call 两者在作用上是相同的，即是调用一个对象的一个方法，以另一个对象替换当前对象。将一个函数的对象上下文从初始的上下文改变

为由 thisObj 指定的新对象。

但两者在参数上有区别的。对于第一个参数意义都一样，但对第二个参数：apply 传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而 call 则作为 call 的参数传入（从第二个参数开始）。如 func.call(func1,var1,var2,var3) 对应的 apply 写法为：func.apply(func1,[var1,var2,var3]) 。

85.原生 JS 的 window.onload 与 Jquery 的$(document).ready(function(){})有什么不同？如何用原生 JS 实现 Jq 的 ready 方法？

window.onload()方法是必须等到页面内包括图片的所有元素加载完毕后才能执行。

$(document).ready()是 DOM 结构绘制完毕后就执行，不必等到加载完毕。

```
/*
 * 传递函数给 whenReady()
 * 当文档解析完毕且为操作准备就绪时，函数作为 document 的方法调用
 */
var whenReady = (function() {                    //这个函数返回 whenReady()函数
    var funcs = [];                  //当获得事件时，要运行的函数
    var ready = false;               //当触发事件处理程序时,切换为 true
    //当文档就绪时,调用事件处理程序
    function handler(e) {
        if(ready) return;            //确保事件处理程序只完整运行一次
        //如果发生 onreadystatechange 事件，但其状态不是 complete 的话,那么文档尚未准备好
```

```
        if(e.type === 'onreadystatechange' && document.readyState !== 'complete') {
            return;
        }
        //运行所有注册函数
        //注意每次都要计算 funcs.length
        //以防这些函数的调用可能会导致注册更多的函数
        for(var i=0; i<funcs.length; i++) {
            funcs[i].call(document);
        }
        //事件处理函数完整执行,切换 ready 状态, 并移除所有函数
        ready = true;
        funcs = null;
    }
    //为接收到的任何事件注册处理程序
    if(document.addEventListener) {
        document.addEventListener('DOMContentLoaded', handler, false);
        document.addEventListener('readystatechange', handler, false); //IE9+
        window.addEventListener('load', handler, false);
    }else if(document.attachEvent) {
```

```
        document.attachEvent('onreadystatechange', handler);
        window.attachEvent('onload', handler);
    }
    //返回 whenReady()函数
    return function whenReady(fn) {
        if(ready) { fn.call(document); }
        else { funcs.push(fn); }
    }
})();
```

如果上述代码十分难懂，下面这个简化版：

```
function ready(fn){
    if(document.addEventListener) {//标准浏览器
        document.addEventListener('DOMContentLoaded', function() {
            //注销事件，避免反复触发
document.removeEventListener('DOMContentLoaded',arguments.callee,
false);
            fn();//执行函数
        }, false);
    }else if(document.attachEvent) {//IE
        document.attachEvent('onreadystatechange', function() {
```

```
                if(document.readyState == 'complete') {
                    document.detachEvent('onreadystatechange',
arguments.callee);
                    fn();//函数执行
                }
            });
        }
};
```

86.（设计题）想实现一个对页面某个节点的拖曳？如何做？（使用原生 JS）

回答出概念即可，下面是几个要点

给需要拖拽的节点绑定 mousedown, mousemove, mouseup 事件

mousedown 事件触发后，开始拖拽

mousemove 时，需要通过 event.clientX 和 clientY 获取拖拽位置，并实时更新位置

mouseup 时，拖拽结束

需要注意浏览器边界的情况

87.数组和对象有哪些原生方法，列举一下？

Array.concat(）连接数组

Array.join(）将数组元素连接起来以构建一个字符串

Array.length 数组的大小

Array.pop(）删除并返回数组的最后一个元素

Array.push(）给数组添加元素

Array.reverse(）颠倒数组中元素的顺序

Array.shift(）将元素移出数组

Array.slice(）返回数组的一部分

Array.sort(）对数组元素进行排序

Array.splice(）插入、删除或替换数组的元素

Array.toLocaleString(）把数组转换成局部字符串

Array.toString(）将数组转换成一个字符串

Array.unshift(）在数组头部插入一个元素

Object.hasOwnProperty(）检查属性是否被继承

Object.isPrototypeOf(）一个对象是否是另一个对象的原型

Object.propertyIsEnumerable(）是否可以通过 for/in 循环看到属性

Object.toLocaleString(）返回对象的本地字符串表示

Object.toString(）定义一个对象的字符串表示

Object.valueOf(）指定对象的原始值

88.JS 怎么实现一个类。怎么实例化这个类

严格来讲 js 中并没有类的概念，不过 js 中的函数可以作为构造函数来使用，通过 new 来实例化，其实函数本身也是一个对象。

89.在 JS 中有哪些会被隐式转换为 false

Undefined、null、关键字 false、NaN、零、空字符串

90.外部 JS 文件出现中文字符，会出现什么问题，怎么解决？

会出现乱码，加 charset=”utf-8”；

91.写一个通用的事件侦听器函数

```
// event(事件)工具集，来源：https://github.com/markyun
markyun.Event = {
    // 页面加载完成后
    readyEvent : function(fn) {
        if (fn==null) {
            fn=document;
        }
        var oldonload = window.onload;
        if (typeof window.onload != 'function') {
            window.onload = fn;
        } else {
            window.onload = function() {
                oldonload();
                fn();
            };
        }
    },
```

```
// 视能力分别使用 dom0||dom2||IE 方式 来绑定事件
// 参数： 操作的元素,事件名称 ,事件处理程序
addEvent : function(element, type, handler) {
    if (element.addEventListener) {
        //事件类型、需要执行的函数、是否捕捉
        element.addEventListener(type, handler, false);
    } else if (element.attachEvent) {
        element.attachEvent('on' + type, function() {
            handler.call(element);
        });
    } else {
        element['on' + type] = handler;
    }
},
// 移除事件
removeEvent : function(element, type, handler) {
    if (element.removeEnentListener) {
        element.removeEnentListener(type, handler, false);
    } else if (element.datachEvent) {
        element.detachEvent('on' + type, handler);
```

```
        } else {
            element['on' + type] = null;
        }
    },
    // 阻止事件（主要是事件冒泡，因为 IE 不支持事件捕获）
    stopPropagation : function(ev) {
        if (ev.stopPropagation) {
            ev.stopPropagation();
        } else {
            ev.cancelBubble = true;
        }
    },
    // 取消事件的默认行为
    preventDefault : function(event) {
        if (event.preventDefault) {
            event.preventDefault();
        } else {
            event.returnValue = false;
        }
    },
```

```
    // 获取事件目标
    getTarget : function(event) {
        return event.target || event.srcElement;
    },
    // 获取 event 对象的引用，取到事件的所有信息，确保随时能使用 event;
    getEvent : function(e) {
        var ev = e || window.event;
        if (!ev) {
            var c = this.getEvent.caller;
            while (c) {
                ev = c.arguments[0];
                if (ev && Event == ev.constructor) {
                    break;
                }
                c = c.caller;
            }
        }
        return ev;
    }
};
```

92.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

{'age':'12', 'name':'back'}

2.DOM 相关

1.原生 DOM 操作和事件相关

1. 如需替换 HTML DOM 中的元素，请使用 replaceChild(newnode,oldnode)方法

2. 从父元素中删除子元素 parent.removeChild(child);

3. insertBefore(newItem,existingItem）在指定的已有子节点之前插入新的子节点

4. appendChild(newListItem 向元素添加新的子节点，作为最后一个子节点

5. document.documentElement – 全部文档 document.body – 文档的主体

http://www.w3school.com.cn/jsref/dom_obj_all.asp

JS 事件：target 与 currentTarget 区别

target 在事件流的目标阶段；currentTarget 在事件流的捕获，目标及冒泡阶段。只有当事件流处在目标阶段的时候，两个的指向才是一样的， 而当处于捕获和冒泡阶段的时候，target 指向被单击的对象而 currentTarget 指向当前事件活动的对象（一般为父级）。

事件模型

事件捕捉阶段：事件开始由顶层对象触发，然后逐级向下传播，直到目标的元素； 处于目标阶段：处在绑定事件的元素上； 事件冒泡阶段：事件由具体的元素先接收，然后逐级向上传播，直到不具体的元素；

阻止冒泡/捕获 event.stopPropagation() 和 IE 的 event.cancelBubble=true

DOM 事件绑定 1.绑定事件监听函数：addEventListener 和 attchEvent 2.在 JavaScript 代码中绑定：获取 DOM 元素 dom.onlick = fn 3.在 DOM 元素中直接绑定：

DOM 事件流包括三个阶段：事件捕获阶段、处于目标阶段、事件冒泡阶段。首先发生的事件捕获，为截获事件提供机会。然后是实际的目标接受事件。最后一个阶段是时间冒泡阶段，可以在这个阶段对事件做出响应。

事件委托

因为事件具有冒泡机制，因此我们可以利用冒泡的原理，把事件加到父级上，触发执行效果。这样做的好处当然就是提高性能了

最重要的是通过 event.target.nodeName 判断子元素

```
        <ul id = "bubble">
            <li>1</li>
            <li>2</li>
            <li>3</li>
            <li>4</li>
        </ul>
    </div>

    window.onload = function () {
        var aUl = document.getElementsById("bubble");
        var aLi = aUl.getElementsByTagName("li");

        //不管在哪个事件中，只要你操作的那个元素就是事件源。
        // ie：window.event.srcElement
        // 标准下:event.target
        aUl.onmouseover = function (ev) {
            var ev = ev || window.event;
            var target = ev.target || ev.srcElement;

            if(target.nodeName.toLowerCase() == "li"){
```

```
                target.style.background = "blue";
            }
        };
    };
```

2.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

3.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

3.BOM 相关

4.JS 高级

1.函数高级

1.闭包

特性：　1.函数嵌套函数　2.函数内部可以引用外部的参数和变量　3.参数和变量不会被垃圾回收机制回收

闭包的缺点就是常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。

为什么要使用闭包：

为了设计私有方法和变量，避免全局变量污染　希望一个变量长期驻扎在内存中

view detail: https://segmentfault.com/a/1190000000652891

2.原型与原型链

当从一个对象那里调取属性或方法时，如果该对象自身不存在这样的属性或方法，就会去自己关联的 prototype 对象那里寻找，如果 prototype 没有，就会去 prototype 关联的前辈 prototype 那里寻找，如果再没有则继续查找 Prototype.Prototype 引用的对象，依次类推，直到

Prototype. ⋯ .Prototype 为 undefined （ Object 的 Prototype 就 是 undefined）从而形成了所谓的“原型链”。

其中 foo 是 Function 对象的实例。而 Function 的原型对象同时又是 Object 的实例。这样就构成了一条原型链。

instanceof 确定原型和实例之间的关系

用来判断某个构造函数的 prototype 属性是否存在另外一个要检测对象的原型链上

对 象 的 __proto__ 指 向 自 己 构 造 函 数 的 prototype 。 obj.__proto__.__proto__... 的 原 型 链 由 此 产 生 ， 包 括 我 们 的 操 作 符 instanceof 正 是 通 过 探 测 obj.__proto__.__proto__... === Constructor.prototype 来验证 obj 是否是 Constructor 的实例。

function C(){}

var o = new C(){}

//true 因为 Object.getPrototypeOf(o) === C.prototype

o instanceof C

instanceof 只能用来判断对象和函数，不能用来判断字符串和数字

isPrototypeOf

用于测试一个对象是否存在于另一个对象的原型链上。

判断父级对象 可检查整个原型链

3.作用域与作用域链

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。

全局函数无法查看局部函数的内部细节，但局部函数可以查看其上层的函数细节，直至全局细节。

当需要从局部函数查找某一属性或方法时，如果当前作用域没有找到，就会上溯到上层作用域查找，

直至全局函数，这种组织形式就是作用域链。

4.apply, call 和 bind 有什么区别?

参考答案：三者都可以把一个函数应用到其他对象上，call、apply 是修改函数的作用域（修改 this 指向），并且立即执行，而 bind 是返回了一个新的函数，不是立即执行．apply 和 call 的区别是 apply 接受数组作为参数，而 call 是接受逗号分隔的无限多个参数列表，

Array.prototype.slice.call(null, args)

function getMax(arr){

```
    return Math.max.apply(null, arr);
}
//call
function foo() {
    console.log(this);//{id: 42}
}

foo.call({ id: 42 });
```

如果该方法是非严格模式代码中的函数，则 null 和 undefined 将替换为全局对象，并且原始值将被包装。 当你调用 apply 传递给它 null 时，就像是调用函数而不提供任何对象

5.谈谈对 this 的理解

this 总是指向函数的直接调用者（而非间接调用者）；

如果有 new 关键字，this 指向 new 出来的那个对象；

在事件中，this 指向触发这个事件的对象，特殊的是，IE 中的 attachEvent 中的 this 总是指向全局对象 Window；

6.那些操作会造成内存泄漏？

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

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。

setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

7.深入贯彻闭包思想，全面理解 JS 闭包形成过程

https://segmentfault.com/a/1190000009886713

8.下面这个 ul，如何点击每一列的时候 alert 其 index?（闭包）

```
<ul id=”test”>
<li>这是第一条</li>
<li>这是第二条</li>
<li>这是第三条</li>
</ul>
// 方法一:
var lis=document.getElementById('2223').getElementsByTagName('li');
for(var i=0;i<3;i++)
{
    lis[i].index=i;
    lis[i].onclick=function(){
        alert(this.index);
```

```
        };
    }
    //方法二:
    var
    lis=document.getElementById('2223').getElementsByTagName('li');
    for(var i=0;i<3;i++){
        lis[i].index=i;
        lis[i].onclick=(function(a){
            return function() {
                alert(a);
            }
        })(i);
    }
```

2.对象高级

1.js 继承方式及其优缺点

原型链继承的缺点

一 是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

2.上下文环境对象
3.new 操作符具体做了什么

1、创建一个空对象，并且 this 变量引用该对象，同时继承了该函数的原型（实例对象通过 __proto__ 属性指向原型对象； obj.__proto__ = Base.prototype;） 2、属性和方法被加入到 this 引用的对象中。

```
function Animal(name) {
    this.name = name;
}

Animal.prototype.run = function() {
    console.log(this.name + 'can run...');
}

var cat = new Animal('cat');
//模拟过程
new Animal('cat')=function(){
    let obj={};  //创建一个空对象
```

obj.__proto__=Animal.prototype;

//把该对象的原型指向构造函数的原型对象，就建立起原型了：obj->Animal.prototype->Object.prototype->null

return Animal.call(obj,'cat');// 绑定 this 到实例化的对象上

}

4.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();
```

3.线程机制

1.同步和异步的区别?

同步：阻塞的

–张三叫李四去吃饭，李四一直忙得不停，张三一直等着，直到李四忙完两个人一块去吃饭

=浏览器向服务器请求数据，服务器比较忙，浏览器一直等着（页面白屏），直到服务器返回数据，浏览器才能显示页面

异步：非阻塞的

–张三叫李四去吃饭，李四在忙，张三说了一声然后自己就去吃饭了，李四

忙完后自己去吃

=浏览器向服务器请求数据，服务器比较忙，浏览器可以自如的干原来的事情（显示页面），服务器返回数据的时候通知浏览器一声，浏览器把返回的数据再渲染到页面，局部更新

5.ES6 相关

1.谈一谈 let 与 var 和 const 的区别?

let 为 ES6 新添加申明变量的命令，它类似于 var，但是有以下不同：

let 命令不存在变量提升，如果在 let 前使用，会导致报错

暂时性死区的本质，其实还是块级作用域必须“先声明后使用”的性质。

let，const 和 class 声明的全局变量不是全局对象的属性。

const 声明的变量与 let 声明的变量类似，它们的不同之处在于，const 声明的变量只可以在声明时赋值，不可随意修改，否则会导致 SyntaxError（语法错误）。

const 只是保证变量名指向的地址不变，并不保证该地址的数据不变。const 可以在多个模块间共享 let 暂时性死区的原因：var 会变量提升，let 不会。

2.箭头函数

箭头函数不属于普通的 function，所以没有独立的上下文。箭头函数体内的 this 对象，就是定义时所在的对象，而不是使用时所在的对象。 由于箭头函

数没有自己的 this，函数对象中的 call、apply、bind 三个方法，无法"覆盖"箭头函数中的 this 值。箭头函数没有原本(传统)的函数有的隐藏 arguments 对象。 箭头函数不能当作 generators 使用，使用 yield 会产生错误。

在以下场景中不要使用箭头函数去定义：

1. 定义对象方法、定义原型方法、定义构造函数、定义事件回调函数。
2. 箭头函数里不但没有 this，也没有 arguments, super ……

3. Map 和 Set

Symbol，Map 和 Set

Map 对象保存键值对。一个对象的键只能是字符串或者 Symbols，但一个 Map 的键可以是任意值。 Set 对象允许你存储任何类型的唯一值，Set 对象是值的集合，Set 中的元素只会出现一次 Symbol 是一种特殊的、不可变的数据类型，可以作为对象属性的标识符使用(Symbol([description]))

```
let mySet = new Set()

mySet.add(1)

mySet.add('hello')

mySet.add('hello')

console.log(mySet.size);//2

console.log(mySet);//Set {1,'hello'}
```

```
//Map 保存键值对也不能有重复的

let myMap = new Map();

let key1 = 'China',key2 = 'America';

myMap.set(key1,'welcome')

myMap.set(key2,'gold bless you')

console.log(myMap);//Map { 'China' => 'welcome', 'America' => 'gold bless you' }

console.log(myMap.get(key1));//welcome

console.log(myMap.get(key2));//gold bless you


let mySymbol = Symbol('symbol1');

let mySymbol2 = Symbol('symbol1');

console.log(mySymbol == mySymbol2);//false

//Symbols 在 for...in 迭代中不可枚举。

let obj = {}

obj['c'] = 'c'

obj.d ='d'

obj[Symbol('a')] = 'a'

obj[Symbol.for('b')] = 'b'

for(let k in obj){
```

```
    console.log(k);//logs 'c' and 'd'
}
```

for...of 可以用来遍历数组，类数组对象，argument，字符串，Map 和 Set，for...in 用来遍历对象

4.Promise 实现原理

现在回顾下 Promise 的实现过程，其主要使用了设计模式中的观察者模式：

通过 Promise.prototype.then 和 Promise.prototype.catch 方法将观察者方法注册到被观察者 Promise 对象中，同时返回一个新的 Promise 对象，以便可以链式调用。

被观察者管理内部 pending、fulfilled 和 rejected 的状态转变，同时通过构造函数中传递的 resolve 和 reject 方法以主动触发状态转变和通知观察者。

Promise.then()是异步调用的，这也是 Promise 设计上规定的，其原因在于同步调用和异步调用同时存在会导致混乱。

为了暂停当前的 promise，或者要它等待另一个 promise 完成，只需要简单地在 then() 函数中返回另一个 promise。

Promise 也有一些缺点。首先，无法取消 Promise，一旦新建它就会立即执行，无法中途取消。其次，如果不设置回调函数，Promise 内部抛出的错误，不会反应到外部。第三，当处于 Pending 状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）。

一般来说，不要在 then 方法里面定义 Reject 状态的回调函数（即 then 的第二个参数），总是使用 catch 方法，理由是更接近同步的写法。 then 的第二个函数参数和 catch 等价

Promise.all 和 Promise.race 的区别？

Promise.all 把多个 promise 实例当成一个 promise 实例,当这些实例的状态都发生改变时才会返回一个新的 promise 实例，才会执行 then 方法。Promise.race 只要该数组中的 Promise 对象的状态发生变化（无论是 resolve 还是 reject）该方法都会返回。

5.Object.is() 与原来的比较操作符“ ===”、“ ==”的区别?

两等号判等，会在比较时进行类型转换;

三等号判等(判断严格)，比较时不进行隐式类型转换,（类型不同则会返回 false）;

Object.is 在三等号判等的基础上特别处理了 NaN 、-0 和 +0 ，保证 -0 和 +0 不再相同，

但 Object.is(NaN, NaN) 会返回 true.

Object.is 应被认为有其特殊的用途,而不能用它认为它比其它的相等对比更宽松或严格。

6.前端面试之 ES6 篇

https://juejin.im/post/59c8aec0f265da065c5e965e

6.跨域

1.JSONP

回调函数+数据就是 JSON With Padding，简单、易部署。（做法：动态插入 script 标签，设置其 src 属性指向提供 JSONP 服务的 URL 地址，查询字符串中加入 callback 指定回调函数，返回的 JSON 被包裹在回调函数中以字符串的形式被返回，需将 script 标签插入 body 底部）。缺点是只支持 GET，不支持 POST（原因是通过地址栏传参所以只能使用 GET）

2.document.domain 跨子域

document.domain 跨子域 （ 例如 a.qq.com 嵌套一个 b.qq.com 的 iframe ，如果 a.qq.com 设置 document.domain 为 qq.com 。b.qq.com 设置 document.domain 为 qq.com， 那么他俩就能互相通信了，不受跨域限制了。 注意：只能跨子域）

3. iframe

window.name + iframe ==> http://www.tuicool.com/articles/viMFbqV，支持跨主域。不支持 POST

4.postMessage()

HTML5 的 postMessage()方法允许来自不同源的脚本采用异步方式进行有限的通信，可以实现跨文本档、多窗口、跨域消息传递。适用于不同窗口 iframe 之间的跨域

5.CORS

CORS（Cross Origin Resource Share）对方服务端设置响应头

设置相应头：”Access-Control-Allow-Origin“

CORS 请求默认不发送 Cookie 和 HTTP 认证信息。如果要把 Cookie 发到服务器，一方面要服务器同意，指定 Access-Control-Allow-Credentials

字段。

6.服务端代理

服务端代理 在浏览器客户端不能跨域访问,而服务器端调用 HTTP 接口只是使用 HTTP 协议，不会执行 JS 脚本，不需要同源策略，也就没有跨越问题。简单地说，就是浏览器不能跨域，后台服务器可以跨域。（一种是通过 http-proxy-middleware 插件设置后端代理;另一种是通过使用 http 模块发出请求）

7.Ajax

1.ajax 请求和原理

```
var xhr = new XMLHTTPRequest();

// 请求 method 和 URI

xhr.open('GET', url);

// 请求内容

xhr.send();

// 响应状态

xhr.status

// xhr 对象的事件响应

xhr.onreadystatechange = function() {}

xhr.readyState

// 响应内容

xhr.responseText
```

AJAX 的工作原理

Ajax 的工作原理相当于在用户和服务器之间加了一个中间层(AJAX 引擎),使用户操作与服务器响应异步化。 Ajax 的原理简单来说通过 XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 javascript 来操作 DOM 而更新页面。

ajax 优缺点

优点：无刷新更新数据 异步与服务器通信 前后端负载均衡

缺点：

1）ajax 干掉了 Back 和 history 功能，对浏览器机制的破坏 2）对搜索引擎支持较弱 3）违背了 URI 和资源定位的初衷

2.fetch 和 Ajax 有什么不同

XMLHttpRequest 是一个设计粗糙的 API，不符合关注分离（Separation of Concerns）的原则，配置和调用方式非常混乱，而且基于事件的异步模型写起来也没有现代的 Promise，generator/yield，async/await 友好。

fetch 是浏览器提供的一个新的 web API，它用来代替 Ajax（XMLHttpRequest），其提供了更优雅的接口，更灵活强大的功能。Fetch 优点主要有：

语法简洁，更加语义化

基于标准 Promise 实现，支持 async/await

```
fetch(url).then(response => response.json())
  .then(data => console.log(data))
  .catch(e => console.log("Oops, error", e))
```

3.GET 和 POST 的区别

GET 和 POST 的区别

GET 使用 URL 或 Cookie 传参，而 POST 将数据放在 BODY 中，这个是因为 HTTP 协议用法的约定。并非它们的本身区别。

GET 方式提交的数据有长度限制，则 POST 的数据则可以非常大，这个是因为它们使用的操作系统和浏览器设置的不同引起的区别。也不是 GET 和 POST 本身的区别。

POST 比 GET 安全，因为数据在地址栏上不可见，这个说法没毛病，但依然不是 GET 和 POST 本身的区别。

GET 和 POST 最大的区别主要是 GET 请求是幂等性的，POST 请求不是。（幂等性：对同一 URL 的多个请求应该返回同样的结果。）因为 get 请求是幂等的，在网络不好的隧道中会尝试重试。如果用 get 请求增数据，会有重复操作的风险，而这种重复操作可能会导致副作用

4.GET,POST,PUT,Delete

1. GET 请求会向数据库获取信息，只是用来查询数据，不会修改，增加数据。使用 URL 传递参数，对所发送的数量有限制，一般在 2000 字符

2. POST 向服务器发送数据，会改变数据的种类等资源，就像 insert 操作

一样，会创建新的内容，大小一般没有限制，POST 安全性高，POST 不会被缓存

3. PUT 请求就像数据库的 update 操作一样，用来修改数据内容，不会增加数据种类

4. Delete 用来删除操作

5.缓存，存储相关（cookie，web storage 和 session）

cookie 优点：1.可以解决 HTTP 无状态的问题，与服务器进行交互 缺点：1.数量和长度限制，每个域名最多 20 条，每个 cookie 长度不能超过 4kb 2.安全性问题。容易被人拦截 3.浪费带宽，每次请求新页面，cookie 都会被发送过去

cookie 和 session 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.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。

sessionStorage 是当前对话的缓存，浏览器窗口关闭即消失，localStorage 持久存在，除非清除浏览器缓存。

页面缓存原理

页面缓存状态是由 http header 决定的，一个浏览器请求信息，一个是服务器响应信息。主要包括 Pragma: no-cache、Cache-Control、 Expires、Last-Modified、If-Modified-Since。

6.什么是 Cookie 隔离？（或者说：请求资源的时候不要让它带 cookie 怎么做）

如果静态文件都放在主域名下，那静态文件请求的时候都带有的 cookie 的数据提交给 server 的，非常浪费流量，

所以不如隔离开。

因为 cookie 有域的限制，因此不能跨域提交请求，故使用非主要域名的时候，请求头中就不会带有 cookie 数据，

这样可以降低请求头的大小，降低请求时间，从而达到降低整体请求延时的目的。

同时这种方式不会将 cookie 传入 Web Server，也减少了 Web Server 对 cookie 的处理分析环节，

提高了 webserver 的 http 请求的解析速度。

7. Ajax 解决浏览器缓存问题？

1、在 ajax 发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。

2、在 ajax 发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。

3、在 URL 后面加上一个随机数： "fresh=" + Math.random();。

4、在 URL 后面加上时间戳："nowtime=" + new Date().getTime();。

5、如果是使用 jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。这样页面的所有 ajax 都会执行这条语句就是不需要保存缓存记录。

8.Node

1.node 核心

核心模块：EventEmitter, Stream, FS, Net 和全局对象

全局对象：process, console, Buffer 和 exports

exports 和 module.exports 区别

exports 是 module.exports 的一个引用 module.exports 初始值为一个空对象 {}，所以 exports 初始值也是 {} require 引用模块后，返回的是 module.exports 而不是 exports

2.单线程优点

Node.js 依托于 v8 引擎，都是以单线程为基础的。单线程资源占用小。单线程避免了传统 PHP 那样频繁创建、切换线程的开销，使执行速度更加迅速

3.Node.js 是如何做到 I/O 的异步和非阻塞的呢

其实 Node 在底层访问 I/O 还是多线程的。Node 可以借助 livuv 来来实现多线程。

如果我们非要让 Node.js 支持多线程，还是提倡使用官方的做法，利用 libuv 库来实现。

cluster 可以用来让 Node.js 充分利用多核 cpu 的性能

4.并行与并发，进程与线程

并发（Concurrent）= 2 队列对应 1 咖啡机.

并行（Parallel）= 2 队列对应 2 咖啡机.

线程是进程下的执行者，一个进程至少会开启一个线程（主线程），也可以开启多个线程。

5.谈谈 Nodejs 优缺点

优点：

1. 事件驱动，异步编程，占用内存少

2. npm 设计得好

缺点：

1. Debug 很困难。没有 stack trace，出了问题很难查找问题的原因；

2 .如果设计不好，很容易让代码充满 callback，代码不优雅；

3. 可靠性低；

4. 单进程，单线程，只支持单核 CPU，不能充分的利用多核 CPU 服务器。

9.模块化

1.模块化开发怎么做?

立即执行函数,不暴露私有成员

```
var module1 = (function(){
    var _count = 0;
    var m1 = function(){
        //...
    };
    var m2 = function(){
        //...
    };
    return {
        m1 : m1,
        m2 : m2
    };
})();
```

（待完善）

2.AMD（Modules/Asynchronous-Definition）、CMD（Common Module Definition）规范区别?

AMD 规范在这里：https://github.com/amdjs/amdjs-api/wiki/AMD

CMD 规范在这里：https://github.com/seajs/seajs/issues/242

Asynchronous Module Definition，异步模块定义，所有的模块将被异步加载，模块加载不影响后面语句运行。所有依赖某些模块的语句均放置在回调函数中。

区别：

1. 对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。不过 RequireJS 从 2.0 开始，也改成可以延迟执行（根据写法不同，处理方式不同）。CMD 推崇 as lazy as possible.

2. CMD 推崇依赖就近，AMD 推崇依赖前置。看代码：

```
// CMD
define(function(require, exports, module) {
    var a = require('./a')
    a.doSomething()
    // 此处略去 100 行
    var b = require('./b') // 依赖可以就近书写
    b.doSomething()
    // ...
})
```

```
// AMD 默认推荐
define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好
    a.doSomething()
    // 此处略去 100 行
    b.doSomething()
    // ...
})
```

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

参考：http://annn.me/how-to-realize-cmd-loader/

四 构建工具

1.webpack 相关问题

1.loader 和 plugin 区别

loader 用于加载某些资源文件，因为 webpack 本身只能打包 CommonJS 规范的 js 文件，对于其他资源，例如 css，图片等，是没有办法加载的，这就需要对应的 loader 将资源转换 plugin 用于扩展 webpack 的功能，直接作用于 webpack，loader 只专注于转换文件，而 plugin 不仅局限于资源加载

Loader 只能处理单一文件的输入输出，而 Plugin 则可以对整个打包过程获得更多的灵活性，譬如 ExtractTextPlugin，它可以将所有文件中的 css 剥离到一个独立的文件中，这样样式就不会随着组件加载而加载了。

2.什么是 chunk

```
// on-demand loaded chunk
const Info = r => require.ensure([], () => r(require('@components/info/')), 'info');
const Index = r => require.ensure([], () => r(require('../general/app/index')), 'index');

const Outer = { template: '<router-view></router-view>' };

const router = new VueRouter({
    mode: 'history',
    routes: [
        {
            path: '/',
            component: Outer,
            children: [
                { path: '', component: Index },
                { path: 'info', component: Info }
            ]
        }
    ]
});
```

Webpack 提供一个功能可以拆分模块，每一个模块称为 chunk，这个功能叫做 Code Splitting。你可以在你的代码库中定义分割点，调用 require.ensure，实现按需加载

3.如何开发一个 loader，原理是啥

A loader is a node module exporting a function.

缓存： Webpack Loader 同样可以利用缓存来提高效率，并且只需在一个可缓存的 Loader 上加一句 this.cacheable() 异步：在一个异步的模块中，回传时需要调用 Loader API 提供的回调方法 this.async()

4.打包原理

webpack 打包，最基本的实现方式，是将所有的模块代码放到一个数组里，通过数组 ID 来引用不同的模块

/**********/

```
/******/ ([
/* 0 */
/***/ function(module, exports, __webpack_require__) {

    __webpack_require__(1);
    __webpack_require__(2);
    console.log('Hello, world!');

/***/ },
/* 1 */
/***/ function(module, exports) {

    var a = 'a.js';
    console.log("I'm a.js");

/***/ },
/* 2 */
/***/ function(module, exports) {

    var b = 'b.js';
```

```
    console.log("I'm b.js");


/***/ }

/******/ ]);
```

可以发现入口 entry.js 的代码是放在数组索引 0 的位置，其它 a.js 和 b.js 的代码分别放在了数组索引 1 和 2 的位置，而 webpack 引用的时候，主要通过__webpack_require__的方法引用不同索引的模块。

5.webpack 和 gulp 的区别

webpack 是一种模块化打包工具，主要用于模块化方案，预编译模块的方案；gulp 是工具链、构建工具，可以配合各种插件做 js 压缩，css 压缩，less 编译 替代手工实现自动化工作。

Grunt/Gulp 更多的是一种工作流；提供集成所有服务的一站式平台； gulp 可以用来优化前端工作流程。

6.如何写一个 plugin

Compiler 在开始打包时就进行实例化，实例对象里面装着与打包相关的环境和参数，包括 options、plugins 和 loaders 等。

compilation 对象，它继承于 compiler，所以能拿到一切 compiler 的内容。Compilation 表示有关模块资源，已编译资源，Compilation 在每次文件变化重新打包时都进行一次实例化

apply 方法：当安装这个插件的时候，这个 apply 方法就会被 webpack compiler 调用。

```
function HelloWorldPlugin(options) {
  // Setup the plugin instance with options...
}

HelloWorldPlugin.prototype.apply = function(compiler) {
  compiler.plugin('done', function() {
    console.log('Hello World!');
  });
};

module.exports = HelloWorldPlugin;
```

7.webpack 打包后文件体积过大怎么办?

很多方法:异步加载模块(代码分割);提取第三方库(使用 cdn 或者 vender);代码压缩；去除不必要的插件；去除 devtool 选项，dllplugin 等等。

五 前端框架

1.jQuery

1.jQuery.fn 的 init 方法返回的 this 指的是什么对象？为什么要返回 this?

this 执行 init 构造函数自身，其实就是 jQuery 实例对象，返回 this 是为了实现 jQuery 的链式操作

2.jquery 中如何将数组转化为 json 字符串，然后再转化回来？

jQuery 中没有提供这个功能，所以你需要先编写两个 jQuery 的扩展：

```
$.fn.stringifyArray = function(array) {
    return JSON.stringify(array)
}

$.fn.parseArray = function(array) {
    return JSON.parse(array)
}
```

然后调用：

```
$("").stringifyArray(array)
```

3.jQuery 的属性拷贝(extend)的实现原理是什么，如何实现深拷贝？

递归赋值

4.jquery.extend 与 jquery.fn.extend 的区别？

* jquery.extend 为 jquery 类添加类方法，可以理解为添加静态方法

* jquery.fn.extend:

源码中 jquery.fn = jquery.prototype，所以对 jquery.fn 的扩展，就是为 jquery 类添加成员函数

使用：

jquery.extend 扩展，需要通过 jquery 类来调用，而 jquery.fn.extend 扩展，所有 jquery 实例都可以直接调用。

5.针对 jQuery 的优化方法？

*基于 Class 的选择性的性能相对于 Id 选择器开销很大，因为需遍历所有 DOM 元素。

*频繁操作的 DOM，先缓存起来再操作。用 Jquery 的链式调用更好。

比如：var str=$("a").attr("href");

*for (var i = size; i < arr.length; i++) {}

for 循环每一次循环都查找了数组（arr）的.length 属性，在开始循环的时候设置一个变量来存储这个数字，可以让循环跑得更快：

for (var i = size, length = arr.length; i < length; i++) {}

6.JQuery 一个对象可以同时绑定多个事件，这是如何实现的？

* 多个事件同一个函数：

```
$("div").on("click mouseover", function(){});
```

* 多个事件不同函数

```
$("div").on({
    click: function(){},
    mouseover: function(){}
});
```

7.bind(), live(), delegate()的区别

bind： 绑定事件，对新添加的事件不起作用，方法用于将一个处理程序附加到每个匹配元素的事件上并返回 jQuery 对象。

live：方法将一个事件处理程序附加到与当前选择器匹配的所有元素(包含现有的或将来添加的）的指定事件上并返回 jQuery 对象。

delegate：方法基于一组特定的根元素将处理程序附加到匹配选择器的所有元素（现有的或将来的）的一个或多个事件上。

8.jQuery 框架中$.ajax()的常用参数有哪些？写一个 post 请求并带有发送数据和返回数据的样例

async 是否异步

url 请求地址

contentType 发送信息至服务器时内容编码类型

data 发送到服务器的数据

dataType 预期服务器返回的数据类型

type 请求类型

success 请求成功回调函数

error 请求失败回调函数

```
$.ajax({
        url: "/jquery/test1.txt",
        type: 'post',
        data: {
```

```
            id: 1
        },
        success: function(data) {
            alert(data);
        }
    }
```

9.jQuery 一个对象可以同时绑定多个事件，这是如何实现的？

jQuery 可以给一个对象同时绑定多个事件，低层实现方式是使用 addEventListner 或 attachEvent 兼容不同的浏览器实现事件的绑定，这样可以给同一个对象注册多个事件。

10.jquery 中如何将数组转化为 json 字符串，然后再转化回来？

jQuery 中没有提供这个功能，所以你需要先编写两个 jQuery 的扩展：

```
$.fn.stringifyArray = function(array) {
    return JSON.stringify(array)
}
$.fn.parseArray = function(array) {
    return JSON.parse(array)
}
```

然后调用：

```
$("").stringifyArray(array)
```

11.Jquery 与 jQuery UI 有啥区别?

jQuery 是操作 dom 的框架，jQueryUI 是基于 jQuery 做的一个 UI 组件库

12.jQuery 和 Zepto 的区别？各自的使用场景?

jQuery 主要用于 pc 端，当然有对应的 jQuerymobile 用于移动端，zepto 比 jQuery 更加小巧，主要用于移动端

13.针对 jQuery 的优化方法?

优先使用 ID 选择器

在 class 前使用 tag(标签名)

给选择器一个上下文

慎用 .live()方法（应该说尽量不要使用）

使用 data()方法存储临时变量

14.Zepto 的点透问题如何解决?

点透主要是由于两个 div 重合，例如：一个 div 调用 show()，一个 div 调用 hide()；这个时候当点击上面的 div 的时候就会影响到下面的那个 div；

解决办法主要有 2 种：

1.github 上有一个叫做 fastclick 的库，它也能规避移动设备上 click 事件的延迟响应，https://github.com/ftlabs/fastclick

将它用 script 标签引入页面(该库支持 AMD，于是你也可以按照 AMD 规范，用诸如 require.js 的模块加载器引入)，并且在 dom ready 时初始化在 body 上，

2.根据分析，如果不引入其它类库，也不想自己按照上述 fastclcik 的思路再开发一套东西，需要 1.一个优先于下面的“divClickUnder”捕获的事件；2.并且通过这个事件阻止掉默认行为（下面的“divClickUnder”对 click 事件的捕获，在 ios 的 safari，click 的捕获被认为和滚屏、点击输入框弹起键盘

等一样，是一种浏览器默认行为，即可以被 event.preventDefault()阻止的行为）。

2.angular

1.使用过 angular 吗？angular 中的过滤器是干什么用的

在表达式中转换数据 姓名为 {{ lastName | uppercase }}

currency，是什么过滤器——格式化数字为货币格式，单位是$符。

3.react

1.react 和 vue 的区别

相同点：

都支持服务端渲染

都有 Virtual DOM，组件化开发，通过 props 参数进行父子组件数据的传递，都实现 webComponents 规范

数据驱动视图

都有支持 native 的方案，React 的 React native，Vue 的 weex

不同点：

React 严格上只针对 MVC 的 view 层，Vue 则是 MVVM 模式

virtual DOM 不一样 vue 会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。而对于 React 而言，每当应用的状态被改变时，全部子组件都会重新渲染。当然，这可以通过 shouldComponentUpdate 这个生命周期方法来进行控制，

组件写法不一样 React 推荐的做法是 JSX + inline style，也就是把 HTML

和 CSS 全都写进 JavaScript 了，即”all in js” Vue 推荐的是使用 webpack + vue-loader 的单文件组件格式，即 html,css,js 写在同一个文件；

数据绑定：Vue 有实现了双向数据绑定，React 数据流动是单向的

state 对象在 react 应用中是不可变的，需要使用 setState 方法更新状态；在 Vue 中，state 对象并不是必须的，数据由 data 属性在 Vue 对象中进行管理。

2.react 的优缺点

优点：

1. 可以通过函数式方法描述视图组件（好处：相同的输入会得到同样的渲染结果，不会有副 作 用；组件不会被实例化，整体渲染性能得到提升）

2. 集成虚拟 DOM（性能好）

3. 单向数据流（好处是更容易追踪数据变化排查问题

4. 一切都是 component：代码更加模块化，重用代码更容易，可维护性高

5. 大量拥抱 es6 新特性

6: jsx

缺点：

1. jsx 的一个问题是，渲染函数常常包含大量逻辑，最终看着更像是程序片段，而不是视觉呈现。后期如果发生需求更改，维护起来工作量将是巨大的

2. 大而全，上手有难度

3.jsx 的优缺点

允许使用熟悉的语法来定义 HTML 元素树 JSX 让小组件更加简单、明了、直观。 更加语义化且易懂的标签 JSX 本质是对 JavaScript 语法的一个扩展，看起来像是某种模板语言，但其实不是。但正因为形似 HTML，描述 UI 就更直观了，也极大地方便了开发； 在 React 中 babel 会将 JSX 转换为 React.createElement 函数调用，然后将 JSX 转换为正确的 JSON 对象（VDOM 也是一个“树”形的结构） React/JSX 乍看之下，觉得非常啰嗦，但使用 JavaScript 而不是模板语法来开发（模板语法比较有局限性），赋予了开发者许多编程能力。

4.dom diff 算法和虚拟 DOM

React 中的 render 方法，返回一个 DOM 描述，结果仅仅是轻量级的 js 对象。Reactjs 只在调用 setState 的时候会更新 dom，而且还是先更新 Virtual Dom，然后和实际 DOM 比较，最后再更新实际 DOM。

React.js 厉害的地方并不是说它比 DOM 快（这句话本来就是错的），而是说不管你数据怎么变化，我都可以以最小的代价来更新 DOM。方法就是我在内存里面用新的数据刷新一个虚拟的 DOM 树，然后新旧 DOM 树进行比较，找出差异，再更新到真正的 DOM 树上。

当我们修改了 DOM 树上一些节点对应绑定的 state，React 会立即将它标记为“脏状态”。在事件循环的最后才重新渲染所有的脏节点。在实际的代码中，会对新旧两棵树进行一个深度优先的遍历，这样每个节点都会有一个唯一的标记，每遍历到一个节点就把该节点和新的的树进行对比。如果有差异的话就记录到一个对象里面，最后把差异应用到真正的 DOM 树上。 算法实现 1 步骤一：用 JS 对象模拟 DOM 树 2 步骤二：比较两棵虚拟 DOM 树的差异 3 步骤三：把差异应用到真正的 DOM 树上 这就是所谓的 diff 算法

dom diff 采用的是增量更新的方式，类似于打补丁。React 需要为数据添加

key 来保证虚拟 DOM diff 算法的效率。key 属性可以帮助 React 定位到正确的节点进行比较，从而大幅减少 DOM 操作次数，提高了性能。

virtual dom，也就是虚拟节点。它通过 JS 的 Object 对象模拟 DOM 中的节点，然后再通过特定的 render 方法将其渲染成真实的 DOM 节点。http://react-china.org/t/dom/638

为什么 js 对象模拟 DOM 会比 js 操作 DOM 来得快

为了解决频繁操作 DOM 导致 Web 应用效率下降的问题，React 提出了“虚拟 DOM”（virtual DOM）的概念。Virtual DOM 是使用 JavaScript 对象模拟 DOM 的一种对象结构。DOM 树中所有的信息都可以用 JavaScript 表述出来，例如：

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

可以用以下 JavaScript 对象来表示：

```
{
  tag: 'ul',
  children: [{
```

```
        tag: 'li', children: ['Item 1'],
        tag: 'li', children: ['Item 2'],
        tag: 'li', children: ['Item 3']
    }]
}
```

这样可以避免直接频繁地操作 DOM，只需要在 js 对象模拟的虚拟 DOM 进行比对，再将更改的部分应用到真实的 DOM 树上

react 组件性能优化

使用 PureRenderMixin 、shouldComponentUpdate 来避免不必要的虚拟 DOM diff，在 render 内部优化虚拟 DOM 的 diff 速度，以及让 diff 结果最小化

5.react 组件间的数据传递

1.兄弟组件不能直接相互传送数据，此时可以将数据挂载在父组件中，由两个组件共享

2.子组件向父组件通讯，可以通过父组件定义事件（回调函数），子组件调用该函数，通过实参的形式来改变父组件的数据来通信

```
//子组件
this.props.onCommentSubmit({author, content, date:new Date().getTime()});
//父组件
```

```
render(){
    return(
      <div className="m-index">
        <div>
          <h1>评论</h1>
        </div>
        <CommentList data={this.state.data} />
        <CommentForm
onCommentSubmit={this.handleCommentSubmit.bind(this)} />
      </div>
    )
}
```

3.非父子组件间的通信：可以使用全局事件来实现组件间的沟通，React 中可以引入 eventProxy 模块，利用 eventProxy.trigger()方法发布消息，eventProxy.on()方法监听并接收消息。

4.组件间层级太深，可以使用上下文方式，让子组件直接访问祖先的数据或函数，通过 this.context.xx

6.无状态组件

无状态组件其实本质上就是一个函数，传入 props 即可，没有 state，也没有生命周期方法。组件本身对应的就是 render 方法。例子如下：

```
function Title({color = 'red', text = '标题'}) {
  let style = {
    'color': color
  }
  return (
    <div style = {style}>{text}</div>
  )
}
```

无状态组件不会创建对象，故比较省内存。没有复杂的生命周期方法调用，故流程比较简单。没有 state，也不会重复渲染。它本质上就是一个函数而已。

对于没有状态变化的组件，React 建议我们使用无状态组件。总之，能用无状态组件的地方，就用无状态组件。

7.高阶组件

高阶组件（HOC）是函数接受一个组件，返回一个新组件。其前身其实是用 ES5 创建组件时可用的 mixin 方法，但是在 react 版本升级过程中，使用 ES6 语法创建组件时，认为 mixin 是反模式，影响了 react 架构组件的封装稳定性，增加了不可控的复杂度，逐渐被 HOC 所替代。 实现高阶组件的方式有：

属性代理

```
import React, { Component } from 'React';

//高阶组件定义
```

```
const HOC = (WrappedComponent) =>
   class WrapperComponent extends Component {
      render() {
         return <WrappedComponent {...this.props} />;
      }
}
//普通的组件
class WrappedComponent extends Component{
      render(){
            //....
      }
}

//高阶组件使用
export default HOC(WrappedComponent)
```

反向继承

反向继承是指返回的组件去继承之前的组件(这里都用 WrappedComponent 代指)

```
const HOC = (WrappedComponent) =>
```

```
class extends WrappedComponent {
  render() {
    return super.render();
  }
}
```

我们可以看见返回的组件确实都继承自 WrappedComponent,那么所有的调用将是反向调用的(例如:super.render()),这也就是为什么叫做反向继承。

8.react 事件和传统事件有什么区别吗

React 实现了一个“合成事件”层（synthetic event system），这个事件模型保证了和 W3C 标准保持一致，所以不用担心有什么诡异的用法，并且这个事件层消除了 IE 与 W3C 标准实现之间的兼容问题。

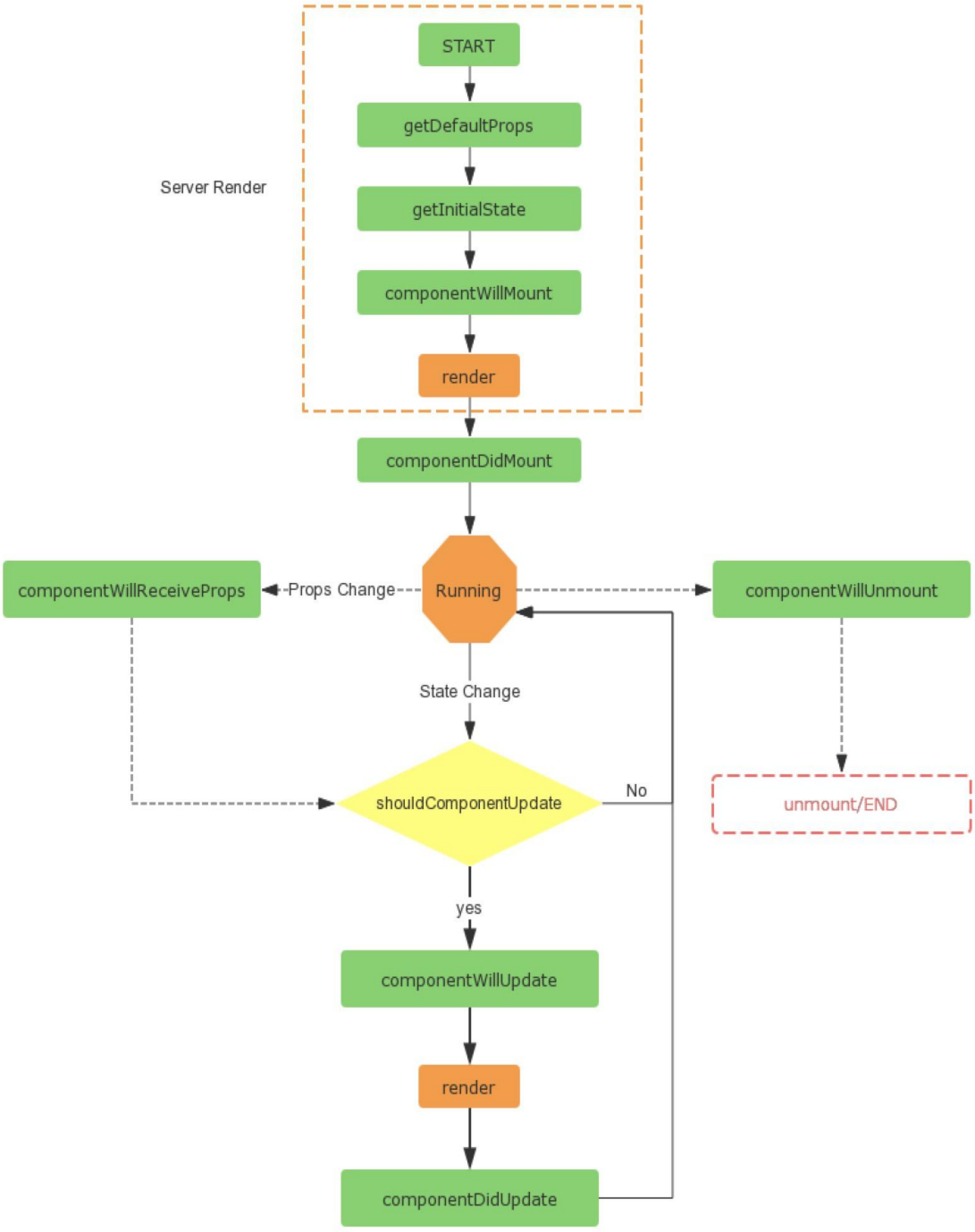
“合成事件”还提供了额外的好处：

事件委托

“合成事件”会以事件委托（event delegation）的方式绑定到组件最上层，并且在组件卸载（unmount）的时候自动销毁绑定的事件。

9.react 的生命周期图

1.组件更新过程



react 组件更新过程：

props/state change：

1.componentWillReceiveProps(nextProps)

只要是父组件的 render 被调用，在 render 中被渲染的子组件就会经历更新的过程。不管父组件传给子组件的 props 有没有改变，都会触发子组件的此函数被调用。注意：通过 setState 方法触发的更新不会调用此函数

2.shouldComponentUpdate(nextProps,nextState)
3.componentWillUpdate 4.render 5.componentDidUpdate

4.vue

1.vue 双向绑定底层实现原理

vue.js 采用数据劫持的方式，结合发布者-订阅者模式，通过 Object.defineProperty()来劫持各个属性的 setter，getter 以监听属性的变动，在数据变动时发布消息给订阅者，触发相应的监听回调：

https://github.com/hawx1993/tech-blog/issues/11

2.vue 虚拟 DOM 和 react 虚拟 DOM 的区别

在渲染过程中，会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。而对于 React 而言，每当应用的状态被改变时，全部子组件都会重新渲染。在 React 应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。如要避免不必要的子组件的重新渲染，你需要在所有可能的地方使用 PureComponent，或是手动实现 shouldComponentUpdate 方法

在 React 中，数据流是自上而下单向的从父节点传递到子节点，所以组件是

简单且容易把握的，子组件只需要从父节点提供的 props 中获取数据并渲染即可。如果顶层组件的某个 prop 改变了，React 会递归地向下遍历整棵组件树，重新渲染所有使用这个属性的组件。

3.v-show 和 v-if 区别

与 v-if 不同的是，无论 v-show 的值为 true 或 false，元素都会存在于 HTML 代码中；而只有当 v-if 的值为 true，元素才会存在于 HTML 代码中

4.vue 组件通信

非父子组件间通信，Vue 有提供 Vuex，以状态共享方式来实现同信，对于这一点，应该注意考虑平衡，从整体设计角度去考量，确保引入她的必要。

父传子: this.$refs.xxx 子传父: this.$parent.xxx

还可以通过$emit 方法出发一个消息，然后$on 接收这个消息

5.你如何评价 vue

框架能够让我们跑的更快，但只有了解原生的 JS 才能让我们走的更远。

vue 专注于 MVVM 中的 viewModel 层，通过双向数据绑定，把 view 层和 Model 层连接了起来。核心是用数据来驱动 DOM。这种把 directive 和 component 混在一起的设计有一个非常大的问题，它导致了很多开发者滥用 Directive（指令），出现了到处都是指令的情况。

优点： 1.不需要 setState，直接修改数据就能刷新页面，而且不需要 react 的 shouldComponentUpdate 就能实现最高效的渲染路径。 2.渐进式的开发模式，模版方式->组件方式->路由整合->数据流整合->服务器渲染。上手的曲线更加平滑简单，而且不像 react 一上来就是组件全家桶 3.v-model

给开发后台管理系统带来极大的便利，反观用 react 开发后台就是个杯具4.html，css 与 js 比 react 更优雅地结合在一个文件上。

缺点：指令太多，自带模板扩展不方便； 组件的属性传递没有 react 的直观和明显

6.说说你对 MVVM 的理解

Model 层代表数据模型，可以在 Model 中定义数据修改和操作业务逻辑；view 代表 Ul 组件。负责将数据模型转换成 Ul 展现出来 ViewModel 是一个同步 View 和 Model 的对象

用户操作 view 层，view 数据变化会同步到 Model，Model 数据变化会立即反应到 view 中。viewModel 通过双向数据绑定把 view 层和 Model 层连接了起来

7.为什么选择 vue

reactjs 的全家桶方式，实在太过强势，而自己定义的 JSX 规范，揉和在 JS 的组件框架里，导致如果后期发生页面改版工作，工作量将会巨大。

vue 的核心：数据绑定 和 视图组件。

Vue 的数据驱动：数据改变驱动了视图的自动更新，传统的做法你得手动改变 DOM 来改变视图，vuejs 只需要改变数据，就会自动改变视图，一个字：爽。再也不用你去操心 DOM 的更新了，这就是 MVVM 思想的实现。

视图组件化：把整一个网页的拆分成一个个区块，每个区块我们可以看作成一个组件。网页由多个组件拼接或者嵌套组成

8.vue 中 mixin 与 extend 区别

全局注册混合对象，会影响到所有之后创建的 vue 实例，而 Vue.extend 是对单个实例进行扩展。

mixin 混合对象（组件复用）

同名钩子函数（bind，inserted，update，componentUpdate，unbind）将混合为一个数组，因此都将被调用，混合对象的钩子将在组件自身钩子之前调用

methods，components，directives 将被混为同一个对象。两个对象的键名（方法名，属性名）冲突时，取组件（而非 mixin）对象的键值对

9.双向绑定和单向数据绑定的优缺点

只有 UI 控件 才存在双向，非 UI 控件 只有单向。 单向绑定的优点是可以带来单向数据流，这样的好处是流动方向可以跟踪，流动单一，没有状态，这使得单向绑定能够避免状态管理在复杂度上升时产生的各种问题，程序的调试会变得相对容易。单向数据流更利于状态的维护及优化，更利于组件之间的通信，更利于组件的复用

双向数据流的优点：

无需进行和单向数据绑定的那些 CRUD（Create，Retrieve，Update，Delete）操作； 双向绑定在一些需要实时反应用户输入的场合会非常方便 用户在视图上的修改会自动同步到数据模型中去，数据模型中值的变化也会立刻同步到视图中去；

缺点：

双向数据流是自动管理状态的，但是在实际应用中会有很多不得不手动处理状态变化的逻辑，使得程序复杂度上升 无法追踪局部状态的变化 双向数据流，值和 UI 绑定，但由于各种数据相互依赖相互绑定，导致数据问题的源头难以被跟踪到

Vue 虽然通过 v-model 支持双向绑定，但是如果引入了类似 redux 的 vuex，就无法同时使用 v-model。

双绑跟单向绑定之间的差异只在于，双向绑定把数据变更的操作隐藏在框架内部，调用者并不会直接感知。

<input v-model="something">

<!-- 等价于以下内容 -->

<input :value="something" @input="something = $event.target.value">

也就是说，你只需要在组件中声明一个 name 为 value 的 props，并且通过触发 input 事件传入一个值，就能修改这个 value。

10.VueJS 开发常见问题集锦

https://segmentfault.com/a/1190000010230843

11.谈谈你对组件的理解

一个组件应该有以下特征：

1. 可组合（Composeable）：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的 2. UI 可以拆分成多个简单的 UI 组件；

3. 可重用（Reusable）：每个组件都是具有独立功能的，它可以被使用在多个 UI 场景；

4. 可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；

5. 可测试（Testable）：因为每个组件都是独立的，那么对于各个组件分别测试显然要比对于整个 UI 进行测试容易的多。

六 移动端问题

1.说说你知道的移动端 web 的兼容性 bug

1、一些情况下对非可点击元素如(label,span)监听 click 事件，ios 下不会触发，css 增加 cursor:pointer 就搞定了。

2.position 在 Safari 下的两个定位需要都写，只写一个容易发生错乱

3.Input 的 placeholder 会出现文本位置偏上的情况

input 的 placeholder 会出现文本位置偏上的情况：PC 端设置 line-height 等于 height 能够对齐，而移动端仍然是偏上，解决是设置 line-height：normal

4.zepto 点击穿透问题

引入 fastclick 解决；event.preventDefault

5.当输入框在最底部的时候，弹起的虚拟键盘会把输入框挡住。

Element.scrollIntoViewIfNeeded(opt_center)

2.有哪些多屏适配方案

1. media query + rem
2. flex
3. 弹性布局
4. flexiable 整体缩放（动态设置缩放系数的方式， 让 layout viewport 与设计图对应，极大地 方便了重构，同时也避免了 1px 的问题）

3.移动端的布局用过媒体查询吗?

假设你现在正用一台显示设备来阅读这篇文章，同时你也想把它投影到屏幕上，或者打印出来， 而显示设备、屏幕投影和打印等这些媒介都有自己的特点，CSS 就是为文档提供在不同媒介上展示的适配方法

当媒体查询为真时，相关的样式表或样式规则会按照正常的级联规被应用。当媒体查询返回假， 标签上带有媒体查询的样式表 仍将被下载 （只不过不会被应用）。

包含了一个媒体类型和至少一个使用 宽度、高度和颜色等媒体属性来限制样式表范围的表达式。CSS3 加入的媒体查询使得无需修改内容便可以使样式应

用于某些特定的设备范围。

<style> @media (min-width: 700px) and (orientation: landscape){ .sidebar { display: none; } } </style>

4.移动端（Android IOS）怎么做好用户体验?

清晰的视觉纵线、

信息的分组、极致的减法、

利用选择代替输入、

标签及文字的排布方式、

依靠明文确认密码、

合理的键盘利用、

5.移动端最小触控区域是多大?

移动端的点击事件的有延迟，时间是多久，为什么会有？ 怎么解决这个延时？（click 有 300ms 延迟,为了实现 safari 的双击事件的设计，浏览器要知道你是不是要双击操作。）

七 大厂面试题

1.美团面试题

事件循环

浏览器中，js 引擎线程会循环从 任务队列 中读取事件并且执行，这种运行机制称作 Event Loop (事件循环).

每个浏览器环境，至多有一个 event loop。 一个 event loop 可以有 1 个或多个 task queue(任务队列)

先执行同步的代码，然后 js 会跑去消息队列中执行异步的代码，异步完成后，再轮到回调函数，然后是去下个事件循环中执行 setTimeout

它从 script(整体代码)开始第一次循环。之后全局上下文进入函数调用栈。直到调用栈清空(只剩全局)，然后执行所有的 micro-task。当所有可执行的 micro-task 执行完毕之后。循环再次从 macro-task 开始，找到其中一个任务队列执行完毕，然后再执行所有的 micro-task，这样一直循环下去。

从规范上来讲，setTimeout 有一个 4ms 的最短时间，也就是说不管你设定多少，反正最少都要间隔 4ms 才运行里面的回调。而 Promise 的异步没有这个问题。Promise 所在的那个异步队列优先级要高一些 Promise 是异步的，是指他的 then()和 catch()方法，Promise 本身还是同步的 Promise 的任务会在当前事件循环末尾中执行，而 setTimeout 中的任务是在下一次事件循环执行

```
//依次输出 12354
setTimeout(function(){
  console.log(4)
  },0);
new Promise(function(resolve){
  console.log(1)
  for( var i=0 ; i<10000 ; i++ ){
    i===9999 && resolve()
  }
```

```
    console.log(2)
}).then(function(){
    console.log(5)
});
console.log(3);
```

xss 和 csrf

事件捕获的应用

jsx 的优点

webpack loader 和 plugin 区别

性能优化

react 和 vue 的区别

vue component 和指令的区别

vue 组件通信

box-sizing

jsonp 缺点，为什么不能用 POST

vue-router 的实现原理

es6 用了哪些新特性

cookie 和 localStorage 区别

git fetch 是干嘛的

事件代理和冒泡，捕获

304 是干嘛的 具体，405 504 又是干嘛的

BFC

其他（自我介绍，为啥离职，为啥从美团离职，git 工作流，code review，单元测试）

react 组件生命周期

伪类和伪元素的区别 CSS 伪类：逻辑上存在但在文档树中却无须标识的“幽灵”分类 CSS 伪元素（:first-letter，:first-line,:after,:before）代表了某个元素的子元素，这个子元素虽然在逻辑上存在，但却并不实际存在于文档树中。CSS3 标准要求伪元素使用双冒号

em 和 rem

2.饿了么面试题

1.什么是类数组对象，如何将类数组对象转为真正的数组

拥有 length 属性和若干索引属性的对象，类数组只有索引值和长度，没有数组的各种方法，所以如果要类数组调用数组的方法，就需要使用 Array.prototype.method.call 来实现。

var arrayLike = {0: 'name', 1: 'age', 2: 'sex', length: 3 }

// 1. slice

Array.prototype.slice.call(arrayLike); // ["name", "age", "sex"]

// 2. splice

Array.prototype.splice.call(arrayLike, 0); // ["name", "age", "sex"]

// 3. ES6 Array.from

Array.from(arrayLike); // ["name", "age", "sex"]

// 4. apply

Array.prototype.concat.apply([], arrayLike)

2.跨域

3.伪元素和伪类

伪类用于当已有元素处于的某个状态时，为其添加对应的样式，这个状态是根据用户行为而动态变化的。

a:link

:first-child

:nth-child

:focus

:visited

伪元素代表了某个元素的子元素，这个子元素虽然在逻辑上存在，但却并不实际存在于文档树中。

4.bind 返回什么

bind() 方法会返回一个新函数，又叫绑定函数，当调用这个绑定函数时，绑定函数会以创建它时传入 bind() 方法的第一个参数作为当前的上下文，即 this，传入 bind() 方法的第二个及之后的参数加上绑定函数运行时自身的参数按照顺序作为原函数的参数来调用原函数.

var x = 8;

```
var o = {
   x: 10,
   getX: function(){
      console.log(this.x);
   }
};
var f = o.getX;
```

f();//8，由于没有绑定执行时的上下文，this 默认指向 window，打印了全局变量 x 的值

```
var g = f.bind(o);
```

g();//10，绑定 this 后，成功的打印了 o 对象的 x 属性的值.

5.git rebase 和 git merge 的区别

merge 操作会生成一个新的节点，之前的提交分开显示。而 rebase 操作不会生成新的节点，是将两个分支融合成一个线性的提交。

6.箭头函数 箭头函数没有它自己的 this 值，箭头函数内的 this 值继承自外围作用域

箭头函数不能用作构造器，不能和 new 一起使用 箭头函数没有原型属性 yield 关键字不能在箭头函数使用 在以下场景中不要使用箭头函数去定义：

定义对象方法、定义原型方法、定义构造函数、定义事件回调函数。
7.== 和===的区别

相等运算符在比较相同类型的数据时，与严格相等运算符完全一样。

在比较不同类型的数据时，相等运算符会先将数据进行类型转换，然后再用严格相等运算符比较。

3.糯米面试题

把面试过程中面试官的问题分享一下：

1. js 有几种数据类型。
2. 将一个字符串转化为数字，将数字转化为字符串。
3. 将数字转化为布尔型。
4. 对数组的几种操作方式。
5. 获取当前网址 url 并从中截取信息。
6. postison 的几种属性
7. rem 移动端
8. 事件委托原理
9. 闭包原理并应用
10. 原型继承的几种方式
11. jsonp 跨域问题，并手写代码
12. 实现斐波那契数列，并手写代码
13. 数组去重，并手写代码

4.2016 百度前端面试题

http://www.cnblogs.com/bhaltair/p/6240418.html

八 扩展性问题

1.职业规划&&未来期望

职业规划：

1.首先应该是一个优秀的程序员

2.其次是努力使自己成为某一领域的技术专家

3.通过技术更好的服务于团队和业务

4.提高沟通能力，团队协作，发现问题，解决问题，总结问题能力

5.写写博客，输出就是最好的学习

6.提升个人前端的工作效率和工作质量

7.关注前端前言技术和发展方向，通过新技术服务团队和业务

8.一专多长

想成为优秀的前端工程师，首先在专业技能领域必不可少，其次在团队贡献、业务思索、价值判断上也有要求。这三方面能决定你的专业技能能够为公司产出多大的价值。

我觉得程序员最核心的竞争力是学习力和责任。 学习能力的源泉就是好奇心，也就是对新知识的渴求，以及对探索未知的冲动。

你希望加入一个什么样的团队
1.对前端开发有激情
2.能够持之以恒的学习
3.团队做事方式是否规范（代码规范，安全规范，流程规范）
4.团队有足够的成长空间，对自己有个清晰的定位。
5.团队认可我的价值

最后你有什么要问我的吗

1.可以问一下公司具体的情况，比如我即将加入的部门的主要业务

2.问一下具体工作情况，比如需要做哪些内容

3.公司的氛围和公司的文化

4.贵司对这项职务的工作内容和期望目标

2.原来公司工作流程是怎么样的，如何与其他人协作的？如何跨部门合作的?

3.你遇到过比较难的技术问题是？你是如何解决的?

4.常使用的库有哪些？常用的前端开发工具？开发过什么应用或组件?

5.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

6.列举 IE 与其他浏览器不一样的特性?

1、事件不同之处:

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;

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;

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；

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()；

7.什么叫优雅降级和渐进增强？

渐进增强 progressive enhancement：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

优雅降级 graceful degradation：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要。降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带。

“优雅降级”观点

“优雅降级”观点认为应该针对那些最高级、最完善的浏览器来设计网站。而将那些被认为“过时”或有功能缺失的浏览器下的测试工作安排在开发周期的最后阶段，并把测试对象限定为主流浏览器（如 IE、Mozilla 等）的前一个版本。

在这种设计范例下，旧版的浏览器被认为仅能提供“简陋却无妨（poor, but passable)” 的浏览体验。你可以做一些小的调整来适应某个特定的浏览器。但由于它们并非我们所关注的焦点，因此除了修复较大的错误之外，其它的差异将被直接忽略。

“渐进增强”观点

“渐进增强”观点则认为应关注于内容本身。

内容是我们建立网站的诱因。有的网站展示它，有的则收集它，有的寻求，有的操作，还有的网站甚至会包含以上的种种，但相同点是它们全都涉及到内容。

这使得“渐进增强”成为一种更为合理的设计范例。这也是它立即被 Yahoo！所采纳并用以构建其“分级式浏览器支持（Graded Browser Support)”策略的原因所在。

8.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。

9.WEB 应用从服务器主动推送 Data 到客户端有那些方式?

html5 提供的 Websocket

不可见的 iframe

WebSocket 通过 Flash

XHR 长时间连接

XHR Multipart Streaming

<script>标签的长时间连接(可跨域)

10.你有用过哪些前端性能优化的方法?

（1） 减少 http 请求次数：CSS Sprites, JS、CSS 源码压缩、图片大小控制合适；网页 Gzip，CDN 托管，data 缓存 ，图片服务器。

（2） 前端模板 JS+数据，减少由于 HTML 标签导致的带宽浪费，前端用变量保存 AJAX 请求结果，每次操作本地变量，不用请求，减少请求次数

（3）用 innerHTML 代替 DOM 操作,减少 DOM 操作次数,优化 javascript 性能。

（4） 当需要设置的样式很多时设置 className 而不是直接操作 style。

（5） 少用全局变量、缓存 DOM 节点查找的结果。减少 IO 读取操作。

（6） 避免使用 CSS Expression（css 表达式）又称 Dynamic properties(动态属性)。

（7） 图片预加载，将样式表放在顶部，将脚本放在底部 加上时间戳。

（8） 避免在页面的主体布局中使用 table，table 要等其中的内容完全下载之后才会显示出来，显示比 div+css 布局慢。

对普通的网站有一个统一的思路，就是尽量向前端优化、减少数据库操作、减少磁盘 IO。向前端优化指的是，在不影响功能和体验的情况下，能在浏览器执行的不要在服务端执行，能在缓存服务器上直接返回的不要到应用服务器，程序能直接取得的结果不要到外部取得，本机内能取得的数据不要到远程取，内存能取到的不要到磁盘取，缓存中有的不要去数据库查询。减少数据库操作指减少更新次数、缓存结果减少查询次数、将数据库执行的操作尽可能的让你的程序完成（例如 join 查询），减少磁盘 IO 指尽量不使用文件系统作为缓存、减少读写文件次数等。程序优化永远要优化慢的部分，换语言是无法“优化”的。

11.一个页面从输入 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：这题胜在区分度高，知识点覆盖广，再不懂的人，也能答出几句，

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。

12.对前端工程师这个职位是怎么样理解的？它的前景会怎么样？

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。

1、实现界面交互

2、提升用户体验

3、有了 Node.js，前端可以实现服务端的一些事情

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。

13.你对加班的看法？

加班就像借钱，原则应当是------救急不救穷

14.平时如何管理你的项目？

先期团队必须确定好全局样式（globe.css），编码模式(utf-8）等；

编写习惯必须一致（例如都是采用继承式的写法，单样式都写成一行）；

标注样式编写人，各模块都及时标注（标注关键样式调用的地方）；

页面进行标注（例如 页面 模块 开始和结束）；

CSS 跟 HTML 分文件夹并行存放，命名都得统一（例如 style.css）；

JS 分文件夹存放 命名以该 JS 功能为准的英文翻译。

图片采用整合的 images.png png8 格式文件使用 尽量整合在一起使用方便将来的管理

15.你在现在的团队处于什么样的角色，起到了什么明显的作用？

16.前端学习网站推荐

1. 极客标签： http://www.gbtags.com/

2. 码农周刊： http://weekly.manong.io/issues/

3. 前端周刊： http://www.feweekly.com/issues

4. 慕课网： http://www.imooc.com/

5. div.io： http://div.io

6. Hacker News： https://news.ycombinator.com/news

7. InfoQ： http://www.infoq.com/

8. w3cplus： http://www.w3cplus.com/

9. Stack Overflow： http://stackoverflow.com/

10.w3school： http://www.w3school.com.cn/

11.mozilla :
https://developer.mozilla.org/zh-CN/docs/Web/JavaScript

17.常使用的库有哪些？常用的前端开发工具？开发过什么应用或组件？

使用率较高的框架有 jQuery、AngularJs, ReactJs, YUI、Prototype、Dojo、Ext.js、Mootools 等。尤其是 jQuery，超过 91%。

轻量级框架有 Modernizr、underscore.js、backbone.js、Raphael.js 等。（理解这些框架的功能、性能、设计原理）

前端开发工具：WebStorm, Hbuilder, Sublime Text 、Eclipse、Notepad、Firebug、HttpWatch、Yslow。

开发过的插件：城市选择插件，汽车型号选择插件、幻灯片插件。弹出层。（写过开源程序，加载器，js 引擎更好）

18.平时如何管理你的项目，如何设计突发大规模并发架构?

先期团队必须确定好全局样式（globe.css），编码模式(utf-8) 等

编写习惯必须一致（例如都是采用继承式的写法，单样式都写成一行）；

标注样式编写人，各模块都及时标注（标注关键样式调用的地方）；

页面进行标注（例如 页面 模块 开始和结束）；

CSS 跟 HTML 分文件夹并行存放，命名都得统一（例如 style.css）

JS 分文件夹存放 命民以该 JS 功能为准英文翻译；

图片采用整合的 images.png png8 格式文件使用 尽量整合在一起使用方便将来的管理

19.那些操作会造成内存泄漏?

先期团队必须确定好全局样式（globe.css），编码模式(utf-8) 等

编写习惯必须一致（例如都是采用继承式的写法，单样式都写成一行）；

标注样式编写人，各模块都及时标注（标注关键样式调用的地方）；

页面进行标注（例如 页面 模块 开始和结束）；

CSS 跟 HTML 分文件夹并行存放，命名都得统一（例如 style.css）

JS 分文件夹存放 命民以该 JS 功能为准英文翻译；

图片采用整合的 images.png png8 格式文件使用 尽量整合在一起使用方便将来的管理

20.你说你热爱前端，那么应该 WEB 行业的发展很关注吧？ 说说最近最流行的一些东西吧?

Node.js、Mongodb、npm、react、angularjs、MVVM、MEAN

九 网络知识相关

1.http 状态码

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）。

]

完整版

1**(信息类)：表示接收到请求并且继续处理

100——客户必须继续发出请求

101——客户要求服务器根据请求转换 HTTP 协议版本

2(响应成功)：表示动作被成功接收、理解和接受**

200——表明该请求被成功地完成，所请求的资源发送回客户端

201——提示知道新文件的 URL

202——接受和处理、但处理未完成

203——返回信息不确定或不完整

204——请求收到，但返回信息为空

205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件

206——服务器已经完成了部分用户的 GET 请求

3(重定向类)：为了完成指定的动作，必须接受进一步处理**

300——请求的资源可在多处得到

301——本网页被永久性转移到另一个 URL

302——请求的网页被转移到一个新的地址，但客户访问仍继续通过原始 URL 地址，重定向，新的 URL 会在 response 中的 Location 中返回，浏览器将会使用新的 URL 发出新的 Request。

303——建议客户访问其他 URL 或访问方式

304——自从上次请求后，请求的网页未修改过，服务器返回此响应时，不会返回网页内容，代表上次的文档已经被缓存了，还可以继续使用

305——请求的资源必须从服务器指定的地址得到

306——前一版本 HTTP 中使用的代码，现行版本中不再使用

307——申明请求的资源临时性删除

4(客户端错误类)：请求包含错误语法或不能正确执行**

400——客户端请求有语法错误，不能被服务器所理解

401——请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用

HTTP 401.1 - 未授权：登录失败

HTTP 401.2 - 未授权：服务器配置问题导致登录失败

HTTP 401.3 - ACL 禁止访问资源

HTTP 401.4 - 未授权：授权被筛选器拒绝

HTTP 401.5 - 未授权：ISAPI 或 CGI 授权失败

402——保留有效 ChargeTo 头响应

403——禁止访问，服务器收到请求，但是拒绝提供服务

HTTP 403.1 禁止访问：禁止可执行访问

HTTP 403.2 – 禁止访问：禁止读访问

HTTP 403.3 – 禁止访问：禁止写访问

HTTP 403.4 – 禁止访问：要求 SSL

HTTP 403.5 – 禁止访问：要求 SSL 128

HTTP 403.6 – 禁止访问：IP 地址被拒绝

HTTP 403.7 – 禁止访问：要求客户证书

HTTP 403.8 – 禁止访问：禁止站点访问

HTTP 403.9 – 禁止访问：连接的用户过多

HTTP 403.10 – 禁止访问：配置无效

HTTP 403.11 – 禁止访问：密码更改

HTTP 403.12 – 禁止访问：映射器拒绝访问

HTTP 403.13 – 禁止访问：客户证书已被吊销

HTTP 403.15 – 禁止访问：客户访问许可过多

HTTP 403.16 – 禁止访问：客户证书不可信或者无效

HTTP 403.17 – 禁止访问：客户证书已经到期或者尚未生效

404——一个 404 错误表明可连接服务器，但服务器无法取得所请求的网页，请求资源不存在。eg：输入了错误的 URL

405——用户在 Request-Line 字段定义的方法不允许

406——根据用户发送的 Accept 拖，请求资源不可访问

407——类似 401，用户必须首先在代理服务器上得到授权

408——客户端没有在用户指定的饿时间内完成请求

409——对当前资源状态，请求不能完成

410——服务器上不再有此资源且无进一步的参考地址

411——服务器拒绝用户定义的 Content-Length 属性请求

412——一个或多个请求头字段在当前请求中错误

413——请求的资源大于服务器允许的大小

414——请求的资源 URL 长于服务器允许的长度

415——请求资源不支持请求项目格式

416——请求中包含 Range 请求头字段，在当前请求资源范围内没有 range 指示值，请求也不包含 If-Range 请求头字段

417——服务器不满足请求 Expect 头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求长。

5(服务端错误类)：服务器不能正确执行一个正确的请求**

HTTP 500 - 服务器遇到错误，无法完成请求

HTTP 500.100 - 内部服务器错误 - ASP 错误

HTTP 500-11 服务器关闭

HTTP 500-12 应用程序重新启动

HTTP 500-13 - 服务器太忙

HTTP 500-14 - 应用程序无效

HTTP 500-15 - 不允许请求 global.asa

Error 501 - 未实现

HTTP 502 - 网关错误

HTTP 503：由于超载或停机维护，服务器目前无法使用，一段时间后可能恢复正常

2.http 报头有哪些

请求头：

Accept

Cache-control

Host

User-agent

Accenp-Language

响应头：

Cache-Control:max-age 避免了服务端和客户端时间不一致的问题。

content-type

Date

Expires

Last-Modified 标记此文件在服务期端最后被修改的时间

httpOnly 设置 cookie 是否能通过 js 去访问。在客户端是不能通过 js 代码去设置一个 httpOnly 类型的 cookie 的，这种类型的 cookie 只能通过服务端来设置。在发生跨域时，cookie 作为一种 credential 信息是不会被传送到服务端的。必须要进行额外设置才可以。

3.代理和反向代理

正向代理： 用浏览器访问时，被残忍的 block，于是你可以在国外搭建一台代理服务器，让代理帮我去请求 google.com，代理把请求返回的相应结构再返回给我。

反向代理：反向代理服务器会帮我们把请求转发到真实的服务器那里去。Nginx 就是性能非常好的反向代理服务器，用来做负载均衡。 正向代理的对象是客户端，反向代理的对象是服务端

4.网络优化/性能优化

使用 CDN，让用户访问最近的资源，减少来回传输时间 合并压缩 CSS、js、图片、静态资源，服务器开启 GZIP css 放顶部，js 放底部（css 可以并行下载，而 js 加载之后会造成阻塞） 图片预加载和首屏图片之外的做懒加载 做 HTTP 缓存（添加 Expires 头和配置 Etag）用户可以重复使用本地缓存，减少对服务器压力 大小超过 10KB 的 css/img 建议外联引用，以细化缓存粒度 小于 10k 的图片 base64 DNS 预解析 DNS-Prefetch 预连接 Preconnect

1.代码层面优化

少用全局变量，减少作用域链查找，缓存 DOM 查找结果，避免使用 with（with 会创建自己的作用域，会增加作用域链长度）；多个变量声明合并；减少 DOM 操作次数；尽量避免在 HTML 标签中写 style 属性

避免使用 css3 渐变阴影效果，尽量使用 css3 动画，开启硬件加速，不滥用 float；避免使用 CSS 表达式；使用<link>来代替@import

2.使用图片预加载技术减少资源请求

5.首屏优化

总结以下可以通过以下三步解决

分拆打包 现在流行的路由库如 react-router 对分拆打包都有很好的支持。可以按照页面对包进行分拆，并在页面切换时加上一些 loading 和 transition 效果。

1.首屏内容最好做到静态缓存 2.首屏内联 css 渲染 3.图片懒加载 4.服务端渲染，首屏渲染速度更快（重点），无需等待 js 文件下载执行的过程 5.交互优化（使用加载占位器，在白屏无法避免的时候，为了解决等待加载过程中白屏或者界面闪烁） 6.图片尺寸大小控制

前端渲染的优势

1. 局部刷新。无需每次都进行完整页面请求

2. 懒加载。如在页面初始时只加载可视区域内的数据，滚动后 rp 加载其它数据，可以通过

3. react-lazyload 实现

4. 富交互。使用 JS 实现各种酷炫效果

5. 节约服务器成本。省电省钱，JS 支持 CDN 部署，且部署极其简单，只需要服务器支持静态文件即可

6. 天生的关注分离设计。服务器来访问数据库提供接口，JS 只关注数据获取和展现

7. JS 一次学习，到处使用。可以用来开发 Web、Serve、Mobile、Desktop 类型的应用

服务端渲染的优势

1. 更好的 SEO，由于搜索引擎爬虫抓取工具可以直接查看完全渲染的页面。

2. 服务端渲染不需要先下载一堆 js 和 css 后才能看到页面（首屏性能）

3. 服务端渲染不用关心浏览器兼容性问题（随意浏览器发展，这个优点逐渐消失）

4. 对于电量不给力的手机或平板，减少在客户端的电量消耗很重要

6.网页验证码是干嘛的，是为了解决什么安全问题。

区分用户是计算机还是人的公共全自动程序。可以防止恶意破解密码、刷票、论坛灌水；

有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试。

7.为什么利用多个域名来存储网站资源会更有效?

CDN 缓存更方便

突破浏览器并发限制

节约 cookie 带宽

节约主域名的连接数，优化页面响应速度

防止不必要的安全问题

8.请谈一下你对网页标准和标准制定机构重要性的理解。

网页标准和标准制定机构都是为了能让 web 发展的更‘健康’，开发者遵循统一的标准，降低开发难度，开发成本，SEO 也会更好做，也不会因为滥用代码导致各种 BUG、安全问题，最终提高网站易用性。

9.前端开发的优化问题。

（1） 减少 http 请求次数：CSS Sprites, JS、CSS 源码压缩、图片大小控制合适；网页 Gzip，CDN 托管，data 缓存 ，图片服务器。

（2） 前端模板 JS+数据，减少由于 HTML 标签导致的带宽浪费，前端用变量保存 AJAX 请求结果，每次操作本地变量，不用请求，减少请求次数

（3）用 innerHTML 代替 DOM 操作，减少 DOM 操作次数，优化 javascript 性能。

（4） 当需要设置的样式很多时设置 className 而不是直接操作 style。

（5） 少用全局变量、缓存 DOM 节点查找的结果。减少 IO 读取操作。

（6） 避免使用 CSS Expression（css 表达式)又称 Dynamic properties(动态属性)。

（7） 图片预加载，将样式表放在顶部，将脚本放在底部　加上时间戳。

（8） 避免在页面的主体布局中使用 table，table 要等其中的内容完全下载之后才会显示出来，显示比 div+css 布局慢。

十 地址收藏

1. 前端面试经典题目合集_牛客网

https://www.nowcoder.com/ta/front-end-interview

2.鹅厂(腾讯)web 前端面试居然还有套路?

https://segmentfault.com/a/1190000009433974

3.面试那些事儿

https://segmentfault.com/a/1190000006950447

4.前沿开发团队的面试过程

https://segmentfault.com/a/1190000006909250

5.向那些年我们经历的一面 N 面致敬

https://segmentfault.com/a/1190000004827689

6.五个典型的 JavaScript 面试题

http://www.zcfy.cc/article/5-typical-javascript-interview-exercises-1960.html

7.征服前端面试

https://segmentfault.com/p/1210000009636596/read

8.阿里面经（附回答）面试必看

http://www.jianshu.com/p/08368a8e4ca7

9.2018 前端校招总结

https://segmentfault.com/a/1190000011635556

10.一道有价值的 JS 继承面试题

https://segmentfault.com/a/1190000008888142

11.当面试官问「你有什么要问我的吗」时，应该问什么?

https://yanzhi.lagou.com/question/281.html

12.最近遇到的前端面试题(2017.02.23 更新版)

http://www.jianshu.com/p/3944732228f0

13.整理下《前端江湖面试》对自己有益的题目。

https://segmentfault.com/a/1190000010868439

14.这里有超过 20 家的前端面试题，你确定不点进来看看?

https://juejin.im/post/58c51b5c44d90400698da686

15.破解前端面试（80% 应聘者不及格系列）

https://juejin.im/post/58f558efac502e006c3e5c97

16.1 月前端面试记

https://juejin.im/post/587dab348d6d810058d87a0a

17.阿里、网易、滴滴共十次前端面试碰到的问题 - 掘金

https://juejin.im/post/59316e682f301e0058378558

18.学生问的一道 javascript 面试题[来自腾讯]

http://www.cnblogs.com/ghostwu/p/7272132.html

19.前端面试必备——基本排序算法

http://blog.csdn.net/owen1190/article/details/76215932

20.技术面试中常被问到是否参与开源，那如何清晰展示 GitHub 项目呢?

http://blog.jobbole.com/112322/

21.你有必要知道的 25 个 JavaScript 面试题

https://segmentfault.com/a/1190000004180569

22.2017 秋招前端工程师百度阿里网易腾讯(全 offer)面经|掘金技术征文

https://juejin.im/post/59cdb971f265da066b397421

23.我的前端进阶之路 - 掘金

https://juejin.im/post/591afe90128fe1005ce339f7

24.前后端分离与前端工程化

https://segmentfault.com/a/1190000006751300

25.大公司里怎样开发和部署前端代码?

https://www.zhihu.com/question/20790576