2-6 面试准备——技术栈准备

技术栈：

- jQuery，核心架构是什么，事件委托是怎么做的，插件机制是什么，（兼容性）；博客文章，按照文章捋顺一次代码即可。
- vue，React，Angular， 准备一到两个，不要全都准备，精心准备一个最好。不要求全都会，但是要对其中一个原理吃透吃深才好。阿里对Vue的源码经常会问。网上看vue源码，1.0较为简洁易懂。找一篇有代码有图的博客文章好好看一下。
- node，尽量不要提，除非有要求。如果说，要说好才行。

关于实战，要多去看社区和论坛，要有自己的项目。

前端工程化工具：
包括环境搭建，预编译，构建，打包等等，具体有sass，less，gulp， browserify， npm，grunt等。
sass和less要准备一下，gulp要准备，npm常用命令，npm的scripts。公司通用webpack，必看必须准备。有一份中文webpack文档非常好。

2-7 面试准备——自我介绍

简历：

- 基本信息，年龄也要写，照片其实不需要
- 学历，写最高的一个即可
- 工作经历，时间-公司-岗位-职责-技术栈-业绩
- 开源项目，GitHub地址

自我陈述：

- 把我面试的沟通方向；学会引导面试官，并且要适可而止，该收住的时候一定要自然的收住
- 豁达、自信的适度发挥；要自信，有气场，不要膨胀，不卑不亢

实例：

- 自如谈兴趣，巧妙示实例，适时讨疑问；兴趣要和前端相关，不要瞎谈，不要硬生生的把面试官拉到某个方向，说半句留半句，让面试官来问。聪明的说法是，当问题回答了一大部分了，但是最后的不会，应该说我得回去思考一下，我这方面确实没有经验，能不能指导一下，我回去再深入学习。
- 节奏要适宜，切忌小聪明；要努力认真的把所有能想到的解决方法都摆出来，不能小聪明的觉得很简单，即使有些方法优缺点，不太好，不常用等等就不去想，不去写，不去思考。要认真的写完每一种解法。

实战：

- 方向要对，过程要细；问题的方向要把握好，不要答非所问，要围绕面试官的出题点去回答，不要瞎答
- 胆子要大，心态要和；不能看到题目难就放弃，不能不去想就放弃，不要怕出错，一定要想。如果问的问题确实不了解，不要自卑，容易导致面试提前终止，也就是不能问什么都不知道，一点儿都不要灰心，要想办法在这个面试中收获点儿什么。可以多向面试官请教一些问题，可以正好找面试官要一些资料去学习，问面试官补完了是不是还可以去面试。知识体系补全了还是可以再次面试的。

3-1 页面布局——一面/二面
面试技巧：

- 准备要充分；基础一定要准备充分
- 知识要系统；前端知识很碎，知识系统了才能答好
- 沟通要简洁；简明扼要，直接针对考察点知识点作答，不要拖泥带水，一语中的即可。
- 内心要诚实；不会就是不会，不要说看过，理解的没那么深，没记住。应该直接说这个方面的知识点确实不了解，向面试官讨教一些学习资料
- 态度要谦虚；不要不把面试官放在眼里，技术实力再强也不行，要谦虚。
- 回答要灵活；很多候选人会根据自己的认知，自己的经验来下结论，这样很不好。技术本身没有好坏，不要自己随便下结论。自己掌握多少就回答多少，不要说xxx就是这样的，或者xxx就不是这样的。

面试模拟：

- 页面布局；最基础的
- CSS盒模型；很基础
- DOM事件；校招必考
- HTTP协议；ws/wss用于直播，不重要。
- 面向对象；
- 原型链；
- 通信；跨域，前后端
- 安全；CSRF，XSS（有老师的免费慕课课程）
- 算法；

页面布局题目：
假设高度已知，请写出三栏布局，其中左栏，右栏宽度各为300px，中间自适应

基础部分是要写出五种解决方法：
float，absolute，flexbox，table（老版本），grid（新技术，会让面试官有好印象）
这种题目看起来简单，但是一定要尽可能多的写出答案才行（至少三种才算及格）

拔高延伸部分：

- 写完之后还要分析各个方法的优缺点，进行比较；浮动的兼容性好，但是会脱离文档流，如果处理不好会导致其他问题。绝对定位最快捷，缺点是脱离了文档流，会导致后续元素全部脱离文档流，会导致这个方案的可使用性较差。flex是移动端最完美的方案，但是ie8以下不支持。

表格布局在历史上评价不高，缺点很多，优点是兼容性很好。网格布局可以做很多复杂的事情，并且代码量非常少，最新的技术，（低版本浏览器的兼容性应该不太好）

- 分析如果高度不已知，哪种依然可以适用；浮动不行，绝对定位不行，flex和table都撑开了两边的高度，能继续使用，grid内容超出了（这都是不改变代码的情况下）。扩展问题有可能问到浮动的解决，BFC等等知识点。
- 每种解决方案的兼容性如何，哪个最实用；

（要去学习flexbox，table和grid的知识）

页面布局小结：

- 语义化掌握到位；不要一路div，要使用语义化标签，学会使用section，article等等
- 页面布局理解深刻；清楚的写出代码
- CSS基础扎实；table，grid，flex等等知识点
- 思维灵活且积极上进；gird是最新的技术，如果没有写出来的话体现不了积极上进，思维要灵活，知道每个方案的优缺点和对比，方案要多才能体现
- 代码书写规范；缩进，类名等等

页面布局的变通：
三栏布局：

- 左右固定，中间自适应；
- 上下固定，中间自适应；（移动端非常常见）

两栏布局：

- 左固定，右自适应；
- 右固定，左自适应；
- 上固定，下自适应；
- 下固定，上自适应；

（这些全都要做一遍，不要眼高手低）

3-4 CSS盒模型——一面/二面
题目：谈谈你对CSS盒模型的认识
—> 基本概念：标准模型+IE模型
—> 标准模型和IE模型的区别
—> CSS如何设置这两种模型
—> JS如何获取和设置盒模型对应的宽高
—> 实例题（根据盒模型解释边距重叠）（拔高题）
—> BFC（边距重叠解决方案）
前面略，第4个问题，获取宽高：

- dom.style.width/height：这种方法只能获得内联的style，（另外还有style节点方式写入样式，通过link链入样式）

- dom.currentStyle.width/height：三种方式的都可以，获得的是渲染之后的页面元素的属性，但是只有IE支持
- window.getComputedStyle(dom).width/height：三种都可以，兼容性更好
- dom.getBoundingClientRect().width/height：计算一个元素的绝对位置，计算一个元素在视窗中的绝对位置，以及宽高（即x，y，width，height四个属性）

第5个问题：

- 父子之间，兄弟之间

第6个问题：BFC（边距重叠解决方案）——块级格式化上下文
—> BFC的基本概念（BFC比IFC常问）
—> BFC的原理（就是渲染规则：1，BFC元素的垂直方向会发生重叠；2，BFC的区域不会与浮动元素的box重叠；3，BFC在页面上是一个独立的容器，外面的元素和里面的元素不会相互影响；4， 计算BFC高度时，浮动元素也会参与计算）
—> 如何创建BFC（1，float值不为none，只要设置了浮动，就创建了BFC；position值不为static和relative就创建了BFC；3，display属性inline-block, table-cell, table等等都可以；4，overflow: hidden/auto（不为visible即可））
—> BFC的使用场景

编程演示：

- 给元素创建一个父元素，创建一个BFC，就可以解决垂直方向边距重叠。
- 元素侵占浮动元素的剩余空间，则在这个元素上面创建BFC，就可以解决，使得BFC不与float重叠。
- 父级元素创建BFC，可以解决由于浮动元素导致父级元素高度为0的情况。创建方法可以是：父级也float，或者父级overflow:hidden/auto

---

3-6 DOM事件——一面/二面

—>基本概念：DOM事件的级别
—> DOM事件模型
—> DOM事件流
—> 描述DOM事件捕获的具体流程
—> Event对象的常见应用
—> 自定义事件

第1个问题：

- DOM0, element.onclick = function(){}
- DOM2, element.addEventListener('click', ()=>{}, false)
- DOM3, element.addEventListener('keyup', ()=>{}, false)

第2个问题：

捕获阶段，目标阶段，冒泡阶段。

第3个问题：
描述DOM事件捕获的具体流程
window --> document —>html —> body —> …… —> target

如何获取html标签，
document.body ==> 获取body
document.documentElement ==> 获取html

第4个问题：

- event.preventDefault()
- event.stopPropagation()
- event.stopImmediatePropagation()
- event.currentTarget
- event.target

第5个问题：

```
// 创建自定义事件
var eve = new Event('custome');
ev.addEventListener('custome', ()=>{})
// 触发自定义事件
ev.dispatch(eve)
```

也可以使用CustomEvent创建自定义实验，并且可以指定一些参数

3-8 HTTP协议类——一面/二面

- HTTP协议的主要特点
- HTTP报文的组成部分
- HTTP方法（校招重点）
- POST和GET的区别
- HTTP状态码（校招重点）
- 什么是持久化连接
- 什么是管线化

第1个问题：
简单快速，灵活，无连接，无状态（后两个一定要说）
简单快速：每个资源都是固定的，统一资源符

灵活：通过一个http协议就可以完成不同数据类型的传输，只需要更改头文件中的数据类型
无连接：连接一次就断掉，不会保持连接
无状态：一次连接之后，服务端是无法记住客户端的状态的（通过session等手段才能实现）

第2个问题：
请求报文：请求行，请求头，空行，请求体
请求行：http方法，页面地址，http协议，版本（example： GET /home/xxx HTTP/1.1）
请求头：key-value值，告诉服务端请求的内容是什么

相应报文：状态行，响应头，空行，响应体
状态行：（example：HTTP/1.1 200 OK）

第3个问题：

- GET —> 获取资源
- POST —> 传输资源
- PUT —> 更新资源
- DELETE —> 删除资源
- HEAD —> 获取报文首部

第4个问题：（记住三四个即可）

- [x] GET在浏览器中回退是无害的，而POST会再次提交请求
- [ ] GET产生的URL地址可以被收藏，而POST不可以
- [x] GET请求可以被浏览器主动缓存，POST不会，需要手动设置
- [ ] GET请求只能进行url编码，而POST支持多种编码方式
- [x] GET请求的参数会被完整的保留在浏览器历史记录里，而POST中的参数不会被保留
- [x] GET请求在URL中传输的参数长度是有限制的，而POST请求没有限制
- [x] 对参数的数据类型，GET只接受ASCII字符，而POST没有限制
- [ ] GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息
- [ ] GET参数通过URL传递，POST放在Request body中

第5个问题：

- 1xx：指示信息
- 2xx：成功
- 3xx：重定向
- 4xx：客户端错误
- 5xx：服务端错误

200，206，301，302，304，400，401，403，404，500，503

第6个问题：
HTTP协议采用“请求-应答”模式，当使用普通模式，即非keep-alive模式时，每次连接都需要重新建立。持久连接只有在1.1版本才支持。

第7个问题：

持久连接的情况下，请求和连接是对应的，即：
请求1--->响应1--->请求2--->响应2

管线化是指，将请求和响应都进行了打包进行发送：
请求1，2，3—>响应1，2，3

- 管线化机制是通过持久连接完成的，至少1.1版本
- 只有GET和HEAD请求可以进行管线化，POST有所限制
- 初次建立不应启动管线化，因为服务端不一定支持1.1
- 管线化不会影响到来的顺序
- 1.1要求服务器支持管线化，但并不要求服务端一定能进行管线化处理，只需要不失败即可
- 开启管线化很可能不会带来大幅度的性能提升，而且很多服务器端和代理程序对管线化的支持并不好，因此现代浏览器，如Chrome和Firefox是默认关闭管线化的。

3-9 原型链——一面/二面

—> 创建对象有几种方法
—> 原型，构造函数，对象实例，原型链
—> instanceof的原理
—> new运算符

- 创建对象的几种方法：
  - 对象字面量：

```
var 01 = {name: 'o1'};
var 011 = new Object({name: 'o11'});
```

  - 构造函数：

```
var M = function(){
  this.name = 'm';
}
var o2 = new M();
```

  - Object.create():

```
var P = {name: 'o3'}
var o3 = Object.create(P);
```

构造函数—new—>实例对象—__proto__—>原型对象
构造函数—prototype—>原型对象
构造函数<—constructor--原型对象

new运算符：
Step1： 一个新对象被创建，它继承自foo.prototype
==>
Step2： 构造函数foo被执行，执行的时候，相应的传参会被传入，同时上下文（this）会被指定为这个新实例。new foo等同于new foo()，只能用在不传递任何参数的情况下
==>
Step3: （如果构造函数返回了一个“对象”，那么这个对象会取代整个new出来的结果。如果构造函数没有返回对象，）那么new出来的结果为步骤1创建的对象。

new的代码：

```
var new2 = function(func) {
    var o = Object.create(func.prototype);
    var k = func.call(o);
    If(typeof k === ‘object’ ) {
        return k;
    } else {
        return o;
    }
}
```

3-11 面向对象——一面/二面

类与实例：怎么生成一个类，类的声明，怎么生成一个实例

类于继承：如何实现继承，继承的几种方式

类的声明：

```
function Animal () {
    this.name = ‘name'
}
```

ES6中的class声明：

```
class Animal2{
    constructor(){
        this.name = ’name'
    }
}
```

实例化类的对象：

```
new Animal()
new Animal2()
```

实现继承的基本原理就是原型链。Js的继承有几种方式：要逐步写。

第1种，借助构造函数实现继承：

```
function Parent1(){
    this.name = ‘parent1’;
}
function child1(){
    Parent1.call(this); //这句话是执行父级的构造函数，也可以用apply，改变函数运行的上下文
    this.type = ‘child1’;
}
new Child1();
```

缺点（问题）：

Parent1的prototype上面的属性或者方法是不能被Chlid1继承的，只实现了部分继承。

第2种，借助原型链实现继承：

```
function Parent2() {
    this.name = ‘parent2’;
}
function Child2() {
    this.type = ‘child2’;
}
Child2.prototype = new Parent2();
new Chlid2();
```

缺点（问题）：

父类中增加新的属性，那么所有的实例都会同时改变。如果某个实例的属性值（来自父类的）发生了变化，那么其他实例也都跟着发生了变化。因为所有实例的__proto__是同一个，所以相互之间会有影响。

第3种，组合方式：

```
function Parent3() {
    this.name = ‘parent3’;
    This.play = [1,2,3,4];
}
function Child3(){
    Parent3.call(this);
    this.type = ‘child3’;
}
Child3.prototype = new Parent3();
var s3 = new Child3();
var s4 = new Child3();
```

缺点（问题）：

父级的构造函数执行了两次，而并没有必要

第4中，组合继承的优化方式1：

```
function Parent4() {
    this.name = ‘parent4’;
    This.play = [1,2,3,4];
}
function Child4(){
    Parent4.call(this);
    this.type = ‘child4’;
}
Child4.prototype = Parent4.prototype;
var s5 = new Child4();
var s6 = new Child4();

s5 instanceof Child4   ======>>>true
s5 instanceof Parent4 ======>>>true
```

问题：
怎么区分一个对象是由子类实例化的，还是父类实例化的（直接实例化）
constructor方法在这里会出问题，因为s5.constructor 指向的是 Parent4。（其实前面的方法也有这个问题）

第5中，组合继承的优化方式2：

```
function Parent5() {
    this.name = ‘parent5’;
    This.play = [1,2,3,4];
}
function Child5(){
    Parent5.call(this);
    this.type = ‘child5’;
}
Child5.prototype = Object.create(Parent5.prototype);
Child5.prototype.constructor = Child5;
var s7 = new Child5();
var s8 = new Child5();
```

面试的时候不建议直接写出最后一种，时间长点儿，就能少问几道，在会的地方多展示一点儿，能提升面试印象。

3-13 通信类——一面/二面

—> 什么是同源策略及限制

—> 前后端如何通信
—> 如何创建Ajax
—> 跨域通信的几种方式

什么是同源策略及限制：
源：协议，域名，端口
限制：不是一个源的文档没有权利去操作另一个源的文档，包括Cookie, LocalStorage, IndexDB, DOM无法获取， Ajax无法发送
同源策略限制不同的源的文档之间进行交互。

前后端如何通信：

- Ajax：同源限制
- WebSocket：不受限制
- CORS：支持同源，也支持非同源（新的通信协议标准）

如何创建Ajax：有以下几个要点

- XMLHttpRequest对象的工作流程
- 兼容性处理
- 事件的触发条件
- 事件的触发顺序

```
var xhr = XMLHttpRequest ? new XMLHttpRequest() :
window.ActiveXObject(‘Microsoft.XMLHTTP’);
var data = opt.data;
var url = opt.url;
var type = opt.type.toUpperCase();
var dataArr = [];
for(k in data){
    dataArr.push(k + ‘=‘ + data[k]);
}
if(type === ‘GET’) {
    url = url + ‘?’ + dataArr.join(‘&’);
    xhr.open(type, url.replace(/\?$/g, ‘’), true);// 如果以？结尾，则将问号去掉
    xhr.send();
}
if(type === ‘POST’) {
    xhr.open(type, url, true);
    xhr.setRequestHeader(‘content-type’, ‘application/x-www-form-urlencoded’);
    xhr.send(dataArr.join('&'))
}
xhr.onload = function() {
    If(xhr.status === 200 || xhr.status === 304) {
```

```
        // 304是使用的本地缓存
        // 206也可以，媒体资源
        var res;
        if(opt.success && opt.success instanceof Function) {
            res = xhr.responseText;
            if(typeof res === 'string'){
                Res = JSON.parse(res);
                opt.success.call(xhr, res);
            }
        } else {
            if(opt.error && ope.error instanceof Function) {
                opt.error.cal(xhr, res);
            }
        }
    }
}
```

跨域通信的几种方式：

- JSONP
- Hash(hash改变，页面是不刷新的，？后是search，改变时会刷新页面)
- postMessage（新技术，h5的标准）
- WebSocket
- CORS（可以理解为支持跨域通信的变种Ajax。当你在浏览器中发送一个ajax跨域请求时，浏览器会在http头中加入一个origin。如果只是一个普通的ajax，则会被浏览器拦截）

JSONP的原理：
利用script标签的可以不同源加载实现的。

1. 在window全局注册一个函数
2. 给服务端传递这个函数的名字，同时可以有参数
3. 服务端返回这个函数，内部填充有数据，就可以拿到数据
4. 删除全局注册的那个函数

Hash的原理：
页面A中通过iframe或frame嵌入了B窗口。目标是A给B发消息。

1. 拿到B的url地址
2. 改变其hash值
3. 在B中接收，onhashchange

WebSocket的原理：

1. var ws = new WebSocket('wss:echo.websocket.org');
2. onopen, onmessage, onclose

CORS的原理：

```
fetch(‘/some/url’, {
    method: ‘get’,
// 加一些配置就可以实现跨域的通信，这里可以参考一篇文章
}).then(function (response) {
    // xxx
}).catch(function(err){
    // yyy
});
```

3-14 安全类——一面/二面

前端安全有几种分类

- XSS
- CSRF

CSRF:

- 基本概念和缩写：跨站请求伪造，cross-site request forgery
- 攻击原理:

用户 ——1登陆——> 网站A
用户 <—— 2下发cookie—— 网站A
用户 —— 3访问——> 网站B
用户 <——4引诱点击—— 网站B
用户 ——5访问——> 网站A（A就会通过cookie判断，觉得是合法用户）

- 防御措施：
  - Token验证：访问网站时，会主动上传cookie，但是不会主动上传token。如果访问接口时，如果没有带token，则不予通过验证。
  - Referer验证：页面来源验证，如果来自于自己站点，则通过，否则不予通过。
  - 隐藏令牌：隐藏在httphead头中，类似于token，只是使用方式的区别

XSS：

- 跨域脚本攻击：Cross-site scripting
- 攻击原理：不需要登陆认证，是向页面注入脚本，写入一些标签。
- 防御措施：目的是让插入的这些东西无法执行

3–15 算法类——一面/二面

- 排序
- 堆栈、队列、链表
- 递归
- 波兰式和逆波兰式

较简单，略
