2019 Sunflower Collection Javascript

**1. javascript的typeof返回哪些数据类型.**

答案：string,boolean,number,undefined,function,object

**2. 例举3种强制类型转换和2种隐式类型转换?**

答案：强制（parseInt,parseFloat,number）  
隐式（== ===）

**3. split() join() 的区别**

答案：前者是将字符串切割成数组的形式，后者是将数组转换成字符串

**4. 数组方法pop() push() unshift() shift()**

答案：push()尾部添加 pop()尾部删除  
unshift()头部添加 shift()头部删除

**5. 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

**6. ajax请求的时候get 和post方式的区别**  
一个在url后面 ，一个放在虚拟载体里面  
get有大小限制(只能提交少量参数)  
安全问题  
应用不同 ，请求数据和提交数据

**7. call和apply的区别**

答案：  
Object.call(this,obj1,obj2,obj3)  
Object.apply(this,arguments)

**8. ajax请求时，如何解析json数据**

答案：使用JSON.parse

**9. 事件委托是什么**

答案: 利用事件冒泡的原理，让自己的所触发的事件，让他的父元素代替执行！

**10. 闭包是什么，有什么特性，对页面有什么影响**

答案：闭包就是能够读取其他函数内部变量的函数,使得函数不被GC回收，如果过多使用闭包，容易导致内存泄露

**11. 如何阻止事件冒泡**

答案：ie:阻止冒泡ev.cancelBubble = true;非IE ev.stopPropagation();

**12. 如何阻止默认事件**

答案：(1)return false；(2) ev.preventDefault();

**13. 添加 删除 替换 插入到某个接点的方法**

答案：

1）创建新节点  
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() //添加  
removeChild() //移除  
replaceChild() //替换  
insertBefore() //插入

3）查找  
getElementsByTagName() //通过标签名称  
getElementsByName() //通过元素的Name属性的值  
getElementById() //通过元素Id，唯一性

**14. 解释jsonp的原理，以及为什么不是真正的ajax**

答案：动态创建script标签，回调函数  
Ajax是页面无刷新请求数据操作

**15. document load 和document ready的区别**

答案：document.onload 是在结构和样式,外部js以及图片加载完才执行js  
document.ready是dom树创建完成就执行的方法，原生种没有这个方法，jquery中有 $().ready(function)

**16. ”==”和“===”的不同**

答案：前者会自动转换类型,再判断是否相等  
后者不会自动类型转换，直接去比较

**17. 函数声明与函数表达式的区别？**

在Javscript中，解析器在向执行环境中加载数据时，对函数声明和函数表达式并非是一视同仁的，解析器会率先读取函数声明，并使其在执行任何代码之前可用（可以访问），至于函数表达式，则必须等到解析器执行到它所在的代码行，才会真正被解析执行。

**18. 对作用域上下文和this的理解，看下列代码：**

var User = {

count: 1,

getCount: function() {

return this.count;

}

};

console.log(User.getCount()); // what?

var func = User.getCount;

console.log(func()); // what?

问两处console输出什么？为什么？

答案:是1和undefined。

　　func是在window的上下文中被执行的，所以不会访问到count属性。

**19. 看下面代码，给出输出结果。**

for(var i = 1; i <= 3; i++){ //建议使用let 可正常输出i的值

setTimeout(function(){

console.log(i);

},0);

};

答案：4 4 4。

原因：Javascript事件处理器在线程空闲之前不会运行。

**20. 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?**

box.onlick= function(){}  
box.addEventListener("click",function(){},false);  
<button onclick="xxx()"></button>

**21. 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

**22. 看下列代码,输出什么?解释原因。**

var a = null;

alert(typeof a);

答案：object

解释：null是一个只有一个值的数据类型，这个值就是null。表示一个空指针对象，所以用typeof检测会返回”object”。

**23. 判断字符串以字母开头，后面可以是数字，下划线，字母，长度为6-30**

var reg=/^[a-zA-Z]\w{5,29}$/;

**24. 回答以下代码，alert的值分别是多少？**

<script>

var a = 100;

function test(){

alert(a);

a = 10; //去掉了var 就变成定义了全局变量了

alert(a);

}

test();

alert(a);

</script>

正确答案是： 100， 10， 10

**25. javaScript的2种变量范围有什么不同？**

全局变量：当前页面内有效

局部变量：函数方法内有效

**26. null和undefined的区别？**

null是一个表示"无"的对象，转为数值时为0；undefined是一个表示"无"的原始值，转为数值时为NaN。

当声明的变量还未被初始化时，变量的默认值为undefined。 null用来表示尚未存在的对象

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

（1）变量被声明了，但没有赋值时，就等于undefined。

（2）调用函数时，应该提供的参数没有提供，该参数等于undefined。

（3）对象没有赋值的属性，该属性的值为undefined。

（4）函数没有返回值时，默认返回undefined。

null表示"没有对象"，即该处不应该有值。典型用法是：

（1） 作为函数的参数，表示该函数的参数不是对象。

（2） 作为对象原型链的终点。

**27. new操作符具体干了什么呢?**

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

2、属性和方法被加入到 this 引用的对象中。

3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

**28. js延迟加载的方式有哪些？**

defer和async、动态创建DOM方式（创建script，插入到DOM中，加载完毕后callBack）、按需异步载入js

**29. Flash、Ajax各自的优缺点，在使用中如何取舍？**

Flash ajax对比

(1)Flash适合处理多媒体、矢量图形、访问机器；对CSS、处理文本上不足，不容易被搜索。

(2)ajax对CSS、文本支持很好，支持搜索；多媒体、矢量图形、机器访问不足。

共同点：与服务器的无刷新传递消息、用户离线和在线状态、操作DOM

**30. 写一个获取非行间样式的函数**

function getStyle(obj,attr) {

if(obj.currentStyle) {

return obj.currentStyle[attr];

}else{

getComputedStyle(obi,false)[attr]

}

}

**31. 希望获取到页面中所有的checkbox怎么做？(不使用第三方框架)**

var inputs = document.getElementsByTagName("input");//获取所有的input标签对象

var checkboxArray = [];//初始化空数组，用来存放checkbox对象。

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

var obj = inputs[i];

if(obj.type=='checkbox'){

checkboxArray.push(obj);

}

}

**32. 写一个function，清除字符串前后的空格。（兼容所有浏览器）**

String.prototype.trim= function(){

return this.replace(/^\s+/,"").replace(/\s+$/,"");

}

**33. javascript语言特性中，有很多方面和我们接触的其他编程语言不太一样,请举例**

javascript语言实现继承机制的核心就是 1 (原型)，而不是Java语言那样的类式继承。Javascript解析引擎在读取一个Object的属性的值时，会沿着 2 (原型链)向上寻找，如果最终没有找到，则该属性值为 3 undefined；如果最终找到该属性的值，则返回结果。与这个过程不同的是，当javascript解析引擎执行“给一个Object的某个属性赋值”的时候，如果当前Object存在该属性，则改写该属性的值，如果当前的Object本身并不存在该属性，则赋值该属性的值。

**34. Cookie在客户机上是如何存储的**

Cookies就是服务器暂存放在你的电脑里的文本文件，好让服务器用来辨认你的计算机。当你在浏览网站的时候，Web服务器会先送一小小资料放在你的计算机上，Cookies 会帮你在网站上所打的文字或是一些选择都记录下来。当下次你再访问同一个网站，Web服务器会先看看有没有它上次留下的Cookies资料，有的话，就会依据Cookie里的内容来判断使用者，送出特定的网页内容给你。

**35. 如何获取javascript三个数中的最大值和最小值？**

Math.max(a,b,c);//最大值

Math.min(a,b,c)//最小值

**36. javascript是面向对象的，怎么体现javascript的继承关系？**

使用prototype原型来实现。

**37. .form中的input可以设置为readonly和disable，请问2者有什么区别？**

readonly不可编辑，但可以选择和复制；值可以传递到后台  
disabled不能编辑，不能复制，不能选择；值不可以传递到后台

**38. 列举javaScript的3种主要数据类型，2种复合数据类型和2种特殊数据类型。**

主要数据类型：string, boolean, number

复合数据类型：function, object

特殊类型：undefined，null

**39. 程序中捕获异常的方法？**

try{

}catch(e){

}finally{

}

**40. Ajax原理**

(1)创建对象

var xhr = new XMLHttpRequest();

(2)打开请求

xhr.open('GET', 'example.txt', true);

(3)发送请求

xhr.send(); 发送请求到服务器

(4)接收响应

xhr.onreadystatechange =function(){}

(1)当readystate值从一个值变为另一个值时，都会触发readystatechange事件。

(2)当readystate==4时，表示已经接收到全部响应数据。

(3)当status ==200时，表示服务器成功返回页面和数据。

(4)如果(2)和(3)内容同时满足，则可以通过xhr.responseText，获得服务器返回的内容。

**41. 解释什么是Json:**

(1)JSON 是一种轻量级的数据交换格式。

(2)JSON 独立于语言和平台，JSON 解析器和 JSON 库支持许多不同的编程语言。

(3)JSON的语法表示三种类型值，简单值(字符串，数值，布尔值，null),数组，对象

**42. js中的3种弹出式消息提醒（警告窗口，确认窗口，信息输入窗口）的命令式什么？**

alert  
confirm  
prompt

**43. 以下代码执行结果**

var uname = 'jack'

function change() {

alert(uname) // ?

var uname = 'lily'

alert(uname) //?

}

change()

分别alert出 undefined，lily，（变量声明提前问题）

**44. 浏览器的滚动距离：**

可视区域距离页面顶部的距离

scrollTop=document.documentElement.scrollTop||document.body.scrollTop

**45. 可视区的大小：**

(1)innerXXX（不兼容ie）

window.innerHeight 可视区高度，包含滚动条宽度

window.innerWidth 可视区宽度，包含滚动条宽度

(2)document.documentElement.clientXXX(兼容ie)

document.documentElement.clientWidth可视区宽度，不包含滚动条宽度

document.documentElement.clientHeight可视区高度，不包含滚动条宽度

**46. 节点的种类有几种，分别是什么？**

(1)元素节点：nodeType ===1;

(2)文本节点：nodeType ===3;

(3)属性节点：nodeType ===2;

**47. innerHTML和outerHTML的区别**

innerHTML(元素内包含的内容）

outerHTML(自己以及元素内的内容）

**48. offsetWidth offsetHeight和clientWidth clientHeight的区别**

(1)offsetWidth （content宽度+padding宽度+border宽度）

(2)offsetHeight（content高度+padding高度+border高度）

(3)clientWidth（content宽度+padding宽度）

(4)clientHeight（content高度+padding高度）

**49. 闭包的好处**

(1)希望一个变量长期驻扎在内存当中(不被垃圾回收机制回收)

(2)避免全局变量的污染

(3)私有成员的存在

(4)安全性提高

**50. 冒泡排序算法**

冒泡排序

var array = [5, 4, 3, 2, 1];

var temp = 0;

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

for (var j = 0; j <array.length - i; j++){

if (array[j] > array[j + 1]){

temp = array[j + 1];

array[j + 1] = array[j];

array[j] = temp;

}

}

**51、js 实现一个函数对javascript中json 对象进行克隆**

var oldObject ="sdf";

var newObject = JSON.parse(JSON.stringify(oldObject));

console.log(newObject);

或者

var a = 'dddd';

function cp(a){return JSON.parse(JSON.stringify(a))}

console.log(cp(a));

**52、js 实现 ajax 请求或者submit请求时 锁屏功能以及开锁功能（请求时界面Loading以及元素不能点击，请求完成即消除Loading）**

function(url, fn) {  
var obj = new XMLHttpRequest(); // XMLHttpRequest对象用于在后台与服务器交换数据  
obj.open('GET', url, true);  
obj.onreadystatechange = function() {  
if(obj.readyState == 4 && obj.status == 200||obj.status == 304) {

loading.style.display = "none"

} else {

alert("不能点击,哈哈哈!");

}

};

obj.send(null);  
}

**53、js 实现一个函数 获得url参数的值**

function getQueryString(name) {

var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)", "i");

var r = window.location.search.substr(1).match(reg);

if (r != null) return unescape(r[2]); return null;

}

**54、请用js计算1-10000中出现的0 的次数**

new Array(10000).fill('').map((\_, index) => index + 1).filter(item => /0/.test(item)).reduce((count, item) => { return count + (String(item).match(/0/g) || []).length}, 0)

**55、写一个function，清除字符串前后的空格。（兼容所有浏览器）**

function trim(str) {

if (str & typeof str === "string") {

return str.replace(/(^s\*)|(s\*)$/g,""); //去除前后空白符

}

}

**56、降维数组**

var arr=[[1,2],[3,4]];

function Jw(obj){

return Array.prototype.concat.apply([],obj);

}

Jw(arr);

**57、将url的查询参数解析成字典对象**

···  
function getQueryObject(url) {  
url = url == null ? window.location.href : url;  
var search = url.substring(url.lastIndexOf("?") + 1);  
var obj = {};  
var reg = /([?&=]+)=([?&=]\*)/g;  
search.replace(reg, function (rs, 2) {  
var name = decodeURIComponent(2);  
val = String(val);  
obj[name] = val;  
return rs;  
});  
return obj;  
}  
···

**58、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');  
···

**59、编写一个方法 求一个字符串的字节长度;**

···  
//假设一个中文占两个字节  
var str = '22两是';

alert(getStrlen(str))

function getStrlen(str){  
var json = {len:0};  
var re = /[\u4e00-\u9fa5]/;  
for (var i = 0; i < str.length; i++) {  
if(re.test(str.charAt(i))){  
json['len']++;  
}  
};  
return json['len']+str.length;  
}  
···

**60、编写一个方法 去掉一个数组的重复元素**

···  
var arr = [1,2,3,1,43,12,12,1];  
var json = {};  
var arr2 = [];  
for (var i = 0; i < arr.length; i++) {  
if(!json[arr[i]]){  
json[arr[i]] = true;  
}else{  
json[arr[i]] = false;  
}

if(json[arr[i]]){

arr2.push(arr[i]);

}

};

for (var i = 0; i < arr.length; i++) {  
if(!aa(arr[i], arr2)){  
arr2.push(arr[i])  
}  
};  
function aa(obj, arr){  
for (var i = 0; i < arr.length; i++) {  
if(arr[i] == obj) return true;  
else return false;  
};  
}  
alert(arr2)  
···

**61、写出3个使用this的典型应用**

事件： 如onclick this->发生事件的对象  
构造函数 this->new 出来的object  
call/apply 改变this

**62、如何深度克隆**

···  
var arr = [1,2,43];  
var json = {a:6,b:4,c:[1,2,3]};  
var str = 'sdfsdf';

var json2 = clone(json);

alert(json['c'])  
function clone(obj){  
var oNew = new obj.constructor(obj.valueOf());  
if(obj.constructor == Object){  
for(var i in obj){  
oNew[i] = obj[i];  
if(typeof(oNew[i]) == 'object'){  
clone(oNew[i]);  
}  
}  
}  
return oNew;  
}  
···

**63、JavaScript中如何检测一个变量是一个String类型？请写出函数实现**

···  
typeof(obj) == 'string'  
obj.constructor == String;  
···

**64、网页中实现一个计算当年还剩多少时间的倒数计时程序，要求网页上实时动态显示“××年还剩××天××时××分××秒”**

···  
var oDate = new Date();  
var oYear = oDate.getFullYear();

var oNewDate = new Date();  
oNewDate.setFullYear(oYear, 11, 31, 23, 59, 59);  
var iTime = oNewDate.getTime()-oDate.getTime();

var iS = iTime/1000;  
var iM = oNewDate.getMonth()-oDate.getMonth();  
var iDate =iS  
···

**65、请解释一下什么是语义化的HTML。**

内容使用特定标签，通过标签就能大概了解整体页面的布局分布

**66、为什么利用多个域名来存储网站资源会更有效？**

确保用户在不同地区能用最快的速度打开网站，其中某个域名崩溃用户也能通过其他郁闷访问网站

**67、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

**68、什么是FOUC？你如何来避免FOUC？**

由于css引入使用了@import 或者存在多个style标签以及css文件在页面底部引入使得css文件加载在html之后导致页面闪烁、花屏  
用link加载css文件，放在head标签里面

**69、文档类型的作用是什么？你知道多少种文档类型？**

影响浏览器对html代码的编译渲染  
html2.0  
xHtml  
html5

**70、浏览器标准模式和怪异模式之间的区别是什么？**

盒模型解释不同

**71、闭包**

子函数能被外部调用到，则该作用连上的所有变量都会被保存下来。

**72、请解释什么是Javascript的模块模式，并举出实用实例。**

js模块化mvc（数据层、表现层、控制层）  
seajs  
命名空间

**73、你如何组织自己的代码？是使用模块模式，还是使用经典继承的方法？**

对内：模块模式  
对外：继承

**74、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

**75、你能解释一下JavaScript中的继承是如何工作的吗？**

子构造函数中执行父构造函数，并用call\apply改变this  
克隆父构造函数原型上的方法

**76、请尽可能详尽的解释AJAX的工作原理。**

创建ajax对象（XMLHttpRequest/ActiveXObject(Microsoft.XMLHttp)）  
判断数据传输方式(GET/POST)  
打开链接 open()  
发送 send()  
当ajax对象完成第四步（onreadystatechange）数据接收完成，判断http响应状态（status）200-300之间或者304（缓存）执行回调函数

**77、最简单的一道题**

···  
var a = 2, b = 3;  
var c = a+++b; // c = 5  
···

**78、var和function的预解析问题,以及变量和function的先后顺序的问题**

···  
// 以下代码执行输出结果是什么  
function b () {  
console.log(a);  
var a = 10;  
function a() {};  
a = 100;  
console.log(a);  
}  
b();

function c () {

console.log(a);

function a() {};

var a = 10;

a = 100;

console.log(a);

}

c();

(function d (num) {

console.log(num);

var num = 10;

}(100))

(function e (num) {

console.log(num);

var num = 10;

function num () {};

}(100))

(function f (num) {

function num () {};

console.log(num);

var num =10

console.log(num);

}(100))

//仍然是预解析(在与解析过程中还要考虑一下当前变量的作用于)

function m () {

console.log(a1); // underfined

console.log(a2); // underfined

console.log(b1); // underfined

console.log(b2); // underfined

if(false) {

function b1 (){};

var a1 = 10;

}

if(true) {

function b2 (){};

var a2 = 10;

}

console.log(a1); // underfined

console.log(a2); // 10

console.log(b1); // underfined

console.log(b2); // function

}

m();

function n() {

if(2>1) {

arr = 10;

brr = 10;

let arr;

var brr;

console.log(arr);

console.log(brr);

}

}

n(); // ReferenceError

···

**79、dom事件委托什么原理，有什么优缺点**

**事件委托原理:事件冒泡机制**

**优点**

1.可以大量节省内存占用，减少事件注册。比如ul上代理所有li的click事件就很不错。  
2.可以实现当新增子对象时，无需再对其进行事件绑定，对于动态内容部分尤为合适

**缺点**

事件代理的常用应用应该仅限于上述需求，如果把所有事件都用事件代理，可能会出现事件误判。即本不该被触发的事件被绑定上了事件。

**80、http的cache机制，以及200状态下怎么实现 from cache（表示接触最多的就是304的from cache）（用于优化，没有接触过，需要理解）**

**含义**

定义：浏览器缓存（Browser Caching）是为了加速浏览，浏览器在用户磁盘上对最近请求过的文档进行存储，当访问者再次请求这个页面时，浏览器就可以从本地磁盘显示文档，这样就可以加速页面的阅览。

**作用**

cache的作用：  
1、减少延迟，让你的网站更快，提高用户体验。  
2、避免网络拥塞，减少请求量，减少输出带宽。

**实现手段**

Cache-Control中的max-age是实现内容cache的主要手段，共有3种常用策略：max-age和Last-Modified（If-Modified-Since）的组合、仅max-age、max-age和ETag的组合。

对于强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。  
对于比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。

**81、一个原型链继承的问题**

// 有一个构造函数A，写一个函数B，继承A

function A (num) {

this.titileName = num;

}

A.prototype = {

fn1: function () {},

fn2: function () {}

}

这个问题的关注点是B继承的A的静态属性，同时B的原型链中不存在A实例的titleName属性

**82、什么是虚拟dom**

React为啥这么大？因为它实现了一个虚拟DOM（Virtual DOM）。虚拟DOM是干什么的？这就要从浏览器本身讲起

如我们所知，在浏览器渲染网页的过程中，加载到HTML文档后，会将文档解析并构建DOM树，然后将其与解析CSS生成的CSSOM树一起结合产生爱的结晶——RenderObject树，然后将RenderObject树渲染成页面（当然中间可能会有一些优化，比如RenderLayer树）。这些过程都存在与渲染引擎之中，渲染引擎在浏览器中是于JavaScript引擎（JavaScriptCore也好V8也好）分离开的，但为了方便JS操作DOM结构，渲染引擎会暴露一些接口供JavaScript调用。由于这两块相互分离，通信是需要付出代价的，因此JavaScript调用DOM提供的接口性能不咋地。各种性能优化的最佳实践也都在尽可能的减少DOM操作次数。

而虚拟DOM干了什么？它直接用JavaScript实现了DOM树（大致上）。组件的HTML结构并不会直接生成DOM，而是映射生成虚拟的JavaScript DOM结构，React又通过在这个虚拟DOM上实现了一个 diff 算法找出最小变更，再把这些变更写入实际的DOM中。这个虚拟DOM以JS结构的形式存在，计算性能会比较好，而且由于减少了实际DOM操作次数，性能会有较大提升

**83、js基础数据类型和引用类型分别是什么？这个前提条件下写一个getType，返回相应的类型**

1.基本数据类型（自身不可拆分的）：Undefined、Null、Boolean、Number、String  
2.引用数据类型（对象）：Object （Array，Date，RegExp，Function）  
ES6基本数据类型多了个symbol 据说这道题刷了百分之二十的人 感谢Abbyshen提出

function gettype(nm){

return Object.prototype.toString.call(nm);

}

**84、 dom选择器优先级是什么，以及权重值计算（一道老问题了）**

1.行内样式 1000  
2.id 0100  
3.类选择器、伪类选择器、属性选择器[type="text"] 0010  
4.标签选择器、伪元素选择器(::first-line) 0001  
5.通配符\*、子选择器、相邻选择器 0000

**85、vue双向数据绑定的原理是什么**

首先传输对象的双向数据绑定 Object.defineProperty(target, key, decription),在decription中设置get和set属性（此时应注意description中get和set不能与描述属性共存）  
数组的实现与对象不同。  
同时运用观察者模式实现wather，用户数据和view视图的更新

**86、 react和vue比较来说有什么区别**

1 component层面，web component和virtual dom  
2 数据绑定（vue双向，react的单向）等好多  
3 计算属性 vue 有，提供方便；而 react 不行  
4 vue 可以 watch 一个数据项；而 react 不行  
5 vue 由于提供的 direct 特别是预置的 directive 因为场景场景开发更容易；react 没有  
6 生命周期函数名太长 directive

**87、git使用过程中，如果你在开发着业务，突然另一个分支有一个bug要改，你怎么办**

git stash //将本次修改存到暂存区（紧急切换分支时）

git stash pop //将所有暂存区的内容取出来

**88、 网页布局有哪几种，有什么区别**

静态、自适应、流式、响应式四种网页布局  
静态布局：意思就是不管浏览器尺寸具体是多少，网页布局就按照当时写代码的布局来布置；  
自适应布局：就是说你看到的页面，里面元素的位置会变化而大小不会变化；  
流式布局：你看到的页面，元素的大小会变化而位置不会变化——这就导致如果屏幕太大或者太小都会导致元素无法正常显示。  
自适应布局：每个屏幕分辨率下面会有一个布局样式，同时位置会变而且大小也会变。

**89、执行下面代码**

var a = {};

var b = {key: 'b'};

var c = {key: 'c'};

var d = [3,5,6];

a[b] = 123;

a[c] = 345;

a[d] = 333;

console.log(a[b]); // 345

console.log(a[c]); // 345

console.log(a[d]); // 333

**90、**

var R = (function() {

var u = {a:1,b:2};

var r = {

fn: function(k) {

return u[k];

}

}

return r;

}());

R.fn('a'); // 1

上述代码中如何获取匿名函数中的u

**91、不适用循环语句（包括map、forEach方法）实现一个100长度的数组，索引值和值相同的数组[0,1,2,3,4,5........99]**

var arr = new Array(100);

//方法1

[...arr.keys()];

//方法二

Array.from(arr.keys());

//方法三

Array.from({length: 100});

// 方法四 借助string

var arr1 = new Array(101);

var str = arr1.join('1,');

str = str.replace(/(1\,)/g, function ($0, $1, index) {

var start = '' + Math.ceil(index/2);

if(index < str.length - 2) {

start += ','

}

return start;

});

return str.split(',');

// 方法五（函数式，参考网络）

function reduce(arr, val) {

if(Object.prototype.toString.apply(val)){

return;

}

if(val >= 100) {

return arr;

}

arr.push(val);

return reduce(arr, val+1);

}

var res = reduce([], 0)

**92、下面语句执行结果输出**

var a = function (val, index) {

console.log(index);

return {

fn: function (name) {

return a(name, val);

}

}

}

var b = a(0); // underfined

b.fn(1); // 0

b.fn(2); // 0

b.fn(3); // 0

**93、科普**

1. dom节点的根节点是不是body  
   回答： 不是，dom节点的根节点是html(包含head和body，head中分为meta、title等。body又分为一组)

2）dom元素都会有offsetParent吗  
回答： offsetParent属性返回一个对象的引用，这个对象是距离调用offsetParent的元素最近的（在包含层次中最靠近的），并且是已进行过CSS定位的容器元素。 如果这个容器元素未进行CSS定位, 则offsetParent属性的取值为根元素(在标准兼容模式下为html元素；在怪异呈现模式下为body元素)的引用。 当容器元素的style.display 被设置为 "none"时（译注：IE和Opera除外），offsetParent属性 返回 null。

1. [1,3,5]转译成字符串是什么  
   回答： '1,3,5'  
   调用toString方法，生成该字符串

4）li标签的祖级元素可以为li，父级元素也可以为例  
回答： 错误

**94、 jsonp原理，jquery是怎么实现的，这样实现有什么好处和坏处**

**原理**

在同源策略下;在某个服务器下的页面是无法获取到该服务器以外的数据的;Jquery中ajax 的核心是通过 XmlHttpRequest获取非本页内容，而jsonp的核心则是动态添加 <script>标签来调用服务器提供的 js脚本  
当我们正常地请求一个JSON数据的时候，服务端返回的是一串 JSON类型的数据，而我们使用 JSONP模式来请求数据的时候服务端返回的是一段可执行的 JavaScript代码。因为jsonp 跨域的原理就是用的动态加载 script的src ，所以我们只能把参数通过 url的方式传递, 所以jsonp的 type类型只能是get ！

$.ajax({

url: 'http://192.168.1.114/yii/demos/test.php', //不同的域

type: 'GET', // jsonp模式只有GET 是合法的

data: {

'action': 'aaron'

},

dataType: 'jsonp', // 数据类型

jsonp: 'backfunc', // 指定回调函数名，与服务器端接收的一致，并回传回来

})

其实jquery 内部会转化成

http://192.168.1.114/yii/demos/test.php?backfunc=jQuery2030038573939353227615\_1402643146875&action=aaron

然后动态加载

<script type="text/javascript"src="http://192.168.1.114/yii/demos/test.php?backfunc=>

然后后端就会执行backfunc(传递参数 )，把数据通过实参的形式发送出去。

在jquery 源码中， jsonp的实现方式是动态添加<script>标签来调用服务器提供的 js脚本。jquery 会在window对象中加载一个全局的函数，当 <script>代码插入时函数执行，执行完毕后就 <script>会被移除。同时jquery还对非跨域的请求进行了优化，如果这个请求是在同一个域名下那么他就会像正常的 Ajax请求一样工作。

**95、 http协议属于七层协议中的哪一层，下一层是什么**

七层结构：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层  
tcp属于传输层；http属于应用层。  
表现层

**96、 js垃圾回收机制知道哪些，v8引擎使用的哪一种**

**js的两种回收机制**

1 标记清除（mark and sweep）  
2 引用计数（reference counting）

**javascript与V8引擎**

垃圾回收机制的好处和坏处

好处：大幅简化程序的内存管理代码，减轻程序猿负担，并且减少因为长时间运转而带来的内存泄露问题。

坏处：自动回收意味着程序猿无法掌控内存。ECMAScript中没有暴露垃圾回收的借口，我们无法强迫其进行垃圾回收，更加无法干预内存管理。

V8 自动垃圾回收算法  
[https://segmentfault.com/a/11...](https://segmentfault.com/a/1190000000440270)

**97、 作用域什么时候生成的？**

页面加载-->创建window全局对象，并生成全局作用域-->然后生成执行上下文，预解析变量(变量提升)，生成全局变量对象；  
$scope

**98、 websocket长连接原理是什么**

**含义**

Websocket是一个持久化的协议，相对于HTTP这种非持久的协议来说。

**原理**

类似长轮循长连接 ; 发送一次请求 ; 源源不断的得到信息

**28. http缓存知道哪些**

[http://blog.csdn.net/yzf91321...](http://blog.csdn.net/yzf913214/article/details/52268651)

**99、 讲一下事件循环机制**

执行上下文(Execution context)  
函数调用栈(call stack)  
队列数据结构(queue)  
Promise

[https://zhuanlan.zhihu.com/p/...](https://zhuanlan.zhihu.com/p/25407758)

**100、 理解web安全吗？都有哪几种，介绍以及如何预防**

1.XSS，也就是跨站脚本注入

攻击方法：

1\. 手动攻击:

编写注入脚本，比如”/><script>alert(document.cookie());</script><!--等，

手动测试目标网站上有的input, textarea等所有可能输入文本信息的区域

2\. 自动攻击

利用工具扫描目标网站所有的网页并自动测试写好的注入脚本，比如：Burpsuite等

防御方法：

1\. 将cookie等敏感信息设置为httponly，禁止Javascript通过document.cookie获得

2\. 对所有的输入做严格的校验尤其是在服务器端，过滤掉任何不合法的输入，比如手机号必须是数字，通常可以采用正则表达式

3\. 净化和过滤掉不必要的html标签，比如：<iframe>, alt,<script> 等

4\. 净化和过滤掉不必要的Javascript的事件标签，比如：onclick, onfocus等

5\. 转义单引号，双引号，尖括号等特殊字符，可以采用htmlencode编码 或者过滤掉这些特殊字符

6\. 设置浏览器的安全设置来防范典型的XSS注入

2.SQL注入

攻击方法：

编写恶意字符串，比如‘ or 1=1--等，

手动测试目标网站上所有涉及数据库操作的地方

防御方法：

1\. 禁止目标网站利用动态拼接字符串的方式访问数据库

2\. 减少不必要的数据库抛出的错误信息

3\. 对数据库的操作赋予严格的权限控制

4\. 净化和过滤掉不必要的SQL保留字，比如：where, or, exec 等

5\. 转义单引号，上引号，尖括号等特殊字符，可以采用htmlencode编码 或者过滤掉这些特殊字符

3.CSRF，也就是跨站请求伪造

就是攻击者冒用用户的名义，向目标站点发送请求

防范方法：

1\. 在客户端进行cookie的hashing，并在服务端进行hash认证

2\. 提交请求是需要填写验证码

3\. 使用One-Time Tokens为不同的表单创建不同的伪随机值

**101、 sessionStorage和localstorage能跨域拿到吗？比如我在**[**www.baidu.com**](http://www.baidu.com)**设置的值能在**[**m.baidu.com**](http://m.baidu.com)**能拿到吗？为什么**

localStorage会跟cookie一样受到跨域的限制，会被document.domain影响

**102、 localstorage不能手动删除的时候，什么时候过期**

除非被清除，否则永久保存 clear()可清楚  
sessionStorage 仅在当前会话下有效，关闭页面或浏览器后被清除

**103、cookie可以设置什么域？可以设置.com吗**

可以通过设置domin来实现

**104、登录状态的保存你认为可以保存在sessionstorage或者localstorage或者cookie或者你知道的哪种方式，存在了哪里？？为什么保存在那里**

**105、 flux -> redux -> mobx 变化的本质是什么**

存储结构 将对象加工可观察 函数式 vs 面向对象  
[https://zhuanlan.zhihu.com/p/...](https://zhuanlan.zhihu.com/p/25585910?refer=purerender)

**106、 按需加载路由怎么加载对应的chunk文件的？换句话说浏览器怎么知道什么时候加载这个chunk，以及webpack是怎么识别那个多个经过hash过的chunk文件**

**107、get和post有什么区别？get可以通过body传递数据吗**

把数据放到 body 里面，必须用 POST 方式取，这是 HTTP 协议限制的。

**108、右边宽度固定，左边自适应**

第一种：

<style>

body{

display: flex;

}

.left{

background-color: rebeccapurple;

height: 200px;

flex: 1;

}

.right{

background-color: red;

height: 200px;

width: 100px;

}

</style>

<body>

<div class="left"></div>

<div class="right"></div>

</body>

第二种

<style>

div {

height: 200px;

}

.left {

float: right;

width: 200px;

background-color: rebeccapurple;

}

.right {

margin-right: 200px;

background-color: red;

}

</style>

<body>

<div class="left"></div>

<div class="right"></div>

</body>

**109、水平垂直居中**

第一种

#container{

position:relative;

}

#center{

width:100px;

height:100px;

position:absolute;

top:50%;

left:50%;

transform: translate(-50%,-50%);

}

第二种

#container{

position:relative;

}

#center{

width:100px;

height:100px;

position:absolute;

top:50%;

left:50%;

margin:-50px 0 0 -50px;

}

第三种

#container{

position:relative;

}

#center{

position:absolute;

margin:auto;

top:0;

bottom:0;

left:0;

right:0;

}

第四种 flex

#container{

display:flex;

justify-content:center;

align-items: center;

}

**109、.四种定位的区别**

static 是默认值  
relative 相对定位 相对于自身原有位置进行偏移，仍处于标准文档流中  
absolute 绝对定位 相对于最近的已定位的祖先元素, 有已定位(指position不是static的元素)祖先元素, 以最近的祖先元素为参考标准。如果无已定位祖先元素, 以body元素为偏移参照基准, 完全脱离了标准文档流。  
fixed 固定定位的元素会相对于视窗来定位,这意味着即便页面滚动，它还是会停留在相同的位置。一个固定定位元素不会保留它原本在页面应有的空隙。

**110、封装一个函数，参数是定时器的时间，.then执行回调函数。**

function sleep (time) {

return new Promise((resolve) => setTimeout(resolve, time));

}

**111、一行代码实现数组去重？**

[...new Set([1,2,3,1,'a',1,'a'])]

**112、使用addEventListener点击li弹出内容，并且动态添加li之后有效**

<ul>

<li>1</li>

<li>2</li>

<li>3</li>

<li>4</li>

</ul>

var ulNode = document.getElementById("ul");

ulNode.addEventListener('click', function (e) {

if (e.target && e.target.nodeName.toUpperCase() == "LI") {

alert(e.target.innerHTML);

}

}, false);

**113、怎么判断两个对象相等**

JSON.stringify(obj)==JSON.stringify(obj);//true

**114、Vue router 除了 router-link 怎么实现跳转?**

router.go(1)

router.push('/')

**115、Vue router 跳转和 location.href 有什么区别？**

router 是 hash 改变  
location.href 是页面跳转，刷新页面

**116、重排和重绘**

部分渲染树（或者整个渲染树）需要重新分析并且节点尺寸需要重新计算。这被称为重排。注意这里至少会有一次重排-初始化页面布局。  
由于节点的几何属性发生改变或者由于样式发生改变，例如改变元素背景色时，屏幕上的部分内容需要更新。这样的更新被称为重绘。

**117、什么情况会触发重排和重绘**

添加、删除、更新 DOM 节点  
通过 display: none 隐藏一个 DOM 节点-触发重排和重绘  
通过 visibility: hidden 隐藏一个 DOM 节点-只触发重绘，因为没有几何变化  
移动或者给页面中的 DOM 节点添加动画  
添加一个样式表，调整样式属性  
用户行为，例如调整窗口大小，改变字号，或者滚动。

**118、js bind 实现机制？手写一个 bind 方法？**

function bind(fn, context){

return function (){

return fn.apply(context, arguments);

}

}

// 柯理化函数思想 感谢pursuitTom的提出

Function.prototype.bind = function (context) {

var args = Array.prototype.slice.call(arguments, 1);

var \_this = this;

return function () {

var thisArgs = [].slice.call(arguments);

return \_this.apply(context, args.concat(thisArgs))

};

}

// ES6写法 感谢waterc的提出

Function.prototype.bind = function(context, ...res) {

let self = this

return function(...arg) {

return self.apply(context, [...res,...arg])

}

}

**119、多个函数**

var a = (function(){return '1';}, function(){return 1;})();

console.log(typeof a); //number

**120、proto、prototype、Object.getPrototypeOf()**

\_\_proto\_\_是指内部原型，和Object.getPrototypeOf()结果等价

function f(){}

f.\_\_proto\_\_ === Object.getPrototypeOf(f); //true

f.prototype === Object.getPrototypeOf(f); //false

**121、浏览记录前后跳转（尚未试验）**

<a href="javascript:history.go(-1)">backward</a>

<a href="javascript:history.go(1)">forward</a>

**122、setTimeout 和 setInterval 细谈**

常问的点，前者是在一定时间过后将函数添加至执行队列，执行时间=延迟时间+之前函数代码执行时间+执行函数时间。  
后者是不管前一次是否执行完毕，每隔一定时间重复执行，用于精准执行互相没有影响的重复操作。  
如果需要控制前后执行顺序，最好使用setTimeout模拟setInterval

var time = 400, times = 0, max = 10;

function func(){

times++;

if(times < max){

//code here

setTimeout(func, time);

} else {

console.log("finished");

}

}

setTimeout(func, time);

**123、判断多图片加载完毕**

注：用jQueryObject.ready()只能判断dom结构加载完毕  
好像描述的不是很清楚，这里写一下代码。  
方法1：

var counter = 0;

var queryInterval = 30; //ms

var pics = document.getElementsByClassName("pics");

function singleQuery(i){

if(pics[i].complete){

counter++;

console.log(i + " is loaded");

} else {

setTimeout(singleQuery, queryInterval, i);

}

}

function allQuery(callback){

if(counter < pics.length){

console.log("current number of loaded pics: " + counter);

setTimeout(allQuery, queryInterval, callback);

} else {

console.log("All pics are loaded.");

callback();

}

}

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

setTimeout(singleQuery, queryInterval, i);

}

setTimeout(allQuery, queryInterval, callback);

主要也是采用setTimeout模拟轮询，判断方式是img标签dom的complete属性（布尔值），缺点是定时器太多。

方法2：

var counter = 0, queryInterval = 50;

var pics = document.getElementsByClassName("pics");

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

pics[i].onload = function(){

counter++;

console.log(this.id + " is loaded");

}

}

function queryPictures(callback){

if(counter < pics.length){

console.log("current number of loaded pics: " + counter);

setTimeout(queryPictures, queryInterval, callback);

} else {

console.log("All pics are loaded");

callback();

}

}

setTimeout(queryPictures, queryInterval, callback);

利用onload绑定图片加载成功后的回调，通过计数器判断是否加载完毕。

**124、CSS margin重叠问题**

块元素在垂直方向上的margin是很奇怪的，会有重叠现象。  
如果display都是block，有三种情况：  
外间距均为正数，竖直方向上会选择最大的外边距作为间隔  
一正一负，间距 = 正 - |负|  
两个负，间距 = 0 - 绝对值最大的那个  
设置display: inline-block的盒子不会有margin重叠，position: absolute的也不会出现。

**125、CSS选择器优先级 && CSS选择器效率**

ID > 类 > 标签 > 相邻 > 子选择器 > 后代选择器 > \* > 属性 > 伪类

**object.propertyIsEnumerable(xxx)**

判断对象中是否有xxx属性，并且能通过for in枚举，如Array对象的length是不可枚举的

**126、判断数组**

function isArray(arr){

return Object.prototype.toString.call(arr) === '[Object Array]';

}

**127、git fetch && git pull**

git pull自动完成了fetch最新远程版本，并且和本地进行merge  
git fetch获得远程分支，要继续手动merge合并

**128、WebSocket**

HTML5带来的新协议，通过类似HTTP的请求建立连接。主要目的是可以获取服务端的推送。  
原来的方式可能是使用long poll（即不中断连接一直等待数据），或者是ajax轮询的方式（每隔一段时间发送请求，建立连接，询问是否有新的数据）。这两种方式的缺点在于long poll的阻塞，以及ajax轮询的冗余连接。  
WebSocket的设计思想有点类似于回调，在发送请求升级服务端的协议并收到确认信息后，服务端一有新的信息/数据就会主动推送给客户端，至于要一次HTTP握手便可以建立持久连接

**129、跨域相关**

只要协议、域名、端口有不同，则视为不同的域。（域名和域名对应的IP也是跨域）

**1.CORS: Cross-Origin Resource Sharing**

基于服务器支持的跨域，服务器设置Access-Control-Allow-Origin响应头，浏览器可允许跨域

**2.设置domain**

能从子域设到主域，如a.b.c.com—>b.c.com—>c.com  
具体情况：在页面中用iframe打开了另一个页面（前提：两个页面主域是相同的）  
利用frameElement.contentWindow.document.domain设置frame子页面的主域，document.domain设置主页面的主域，之后就能互相获取dom中的数据。  
缺点是只能用于不同子域间的交互。

**3.例如拥有src属性的img标签，每次改变src属性，都会发起http请求。**

var img = new Image();

img.onload = function(){

//code here

};

img.onerror = function(){

//code here

};

img.src="http://server.com/data?query=3";

缺点是只能使用GET请求，不能获取数据，一般用于提交统计信息什么的。  
script、link、iframe只有在添加到DOM中才会发起请求

**4.HTML5 postMessage**

支持IE8+和主流浏览器，写法也简单..

//source: http://test.org:4000

//get the window object of target origin

var win = window.open("http://target.com");

//or this, when a frame is used

var win = document.getElementById("targetId").contentWindow;

win.postMessage("data here", "http://target.com/rest");

//target: http://target.com/tiny

function handleMessage(event){

if(event.orgin!="http://test.org:4000")

return;

var data = event.data;

//code here

//event.source is window.opener

event.source.postMessage("response data here", event.origin);

}

window.addEventListener("message", handleMessage, false);

**5.window.name**

即使在页面打开多层iframe后，每个iframe中window.name 属性值都是相同的，以此用作数据传输的工具。  
但由于跨域的限制，是无法获取另一个frame中的window.name数据，所以要使用一个同域的代理(proxy.html)：

**6. jsonp**

目前主流跨域方法  
调用其他域的脚本获取数据，前提是另一个域能知道回调函数名，这个可以通过请求发送给目标域。  
直接写jQuery封的jsonp

$.getJSON("http://target.com/data?callback=callbackFunctionName", function(data){});

$.getJSON会把获取的responseText转为json，如果url中有callback参数，数据会以script标签形式获取。

**130、闭包相关**

**什么是闭包**

闭包是指有权访问另一个函数作用域中变量的函数

**怎么创建闭包**

在函数内部嵌套使用函数

function fn() {

for (var i = 0; i < 2; i++) {

(function () {

var variate = i;

setTimeout(function () {

console.log("setTimeout执行后:"+variate);

}, 1000);

})();//闭包,立即执行函数,匿名函数

}

console.log(i);//2

console.log(variate);//variate is not defined

}

fn();

**为什么用闭包**

* 因为在闭包内部保持了对外部活动对象的访问,但外部的变量却无法直接访问内部,避免了全局污染;
* 可以当做私有成员,弥补了因js语法带来的面向对象编程的不足;
* 可以长久的在内存中保存一个自己想要保存的变量.

**闭包的缺点**

* 可能导致内存占用过多,因为闭包携带了自身的函数作用域
* 闭包只能取得外部包含函数中得最后一个值

详见[https://segmentfault.com/a/11...](https://segmentfault.com/a/1190000014540734?utm_source=channel-hottest)

**131、a:active 移动端实现**

有时候一些按钮的简单点击交互可以通过css伪类来实现；必须点击了更改颜色；松开恢复；IOS手机会出现伪类无效的情况；iOS系统的移动设备中，需要在按钮元素或body/html上绑定一个touchstart事件才能激活:active状态。

document.body.addEventListener('touchstart', function () { //...空函数即可});

**132、ios滑动卡顿**

-webkit-overflow-scrolling:touch 可能会在IOS系统低的情况出现滚动条；尝试溢出解决

**133、forEach和map的区别**

相同点

* 都是循环遍历数组中的每一项
* forEach和map方法里每次执行匿名函数都支持3个参数，参数分别是item（当前每一项）、index（索引值）、arr（原数组）
* 匿名函数中的this都是指向window
* 只能遍历数组
* 都有兼容问题

不同点

* map速度比foreach快
* map会返回一个新数组，不对原数组产生影响,foreach不会产生新数组，
* map因为返回数组所以可以链式操作，foreach不能

**134、浅拷贝和深拷贝**

jQuery.extend第一个参数可以是布尔值，用来设置是否深度拷贝的

jQuery.extend(true, { a : { a : "a" } }, { a : { b : "b" } } );

jQuery.extend( { a : { a : "a" } }, { a : { b : "b" } } );

最简单的深拷贝

aa = JSON.parse( JSON.stringify(a) )

浅复制--->就是将一个对象的内存地址的“”编号“”复制给另一个对象。深复制--->实现原理，先新建一个空对象，内存中新开辟一块地址，把被复制对象的所有可枚举的(注意可枚举的对象)属性方法一一复制过来，注意要用递归来复制子对象里面的所有属性和方法，直到子子.....属性为基本数据类型。总结，深复制理解两点，1,新开辟内存地址，2,递归来刨根复制。

**135、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。

**136、js加载位置区别优缺点**

html文件是自上而下的执行方式，但引入的css和javascript的顺序有所不同，css引入执行加载时，程序仍然往下执行，而执行到<script>脚本是则中断线程，待该script脚本执行结束之后程序才继续往下执行。  
所以，大部分网上讨论是将script脚本放在<body>之后，那样dom的生成就不会因为长时间执行script脚本而延迟阻塞，加快了页面的加载速度。

但又不能将所有的script放在body之后，因为有一些页面的效果的实现，是需要预先动态的加载一些js脚本。所以这些脚本应该放在<body>之前。

其次，不能将需要访问dom元素的js放在body之前，因为此时还没有开始生成dom，所以在body之前的访问dom元素的js会出错，或者无效

**script放置位置的原则“页面效果实现类的js应该放在body之前，动作，交互，事件驱动，需要访问dom属性的js都可以放在body之后**