# 第三阶段面试题

## 一、JavaScript高级

### 1. 判断以下程序的输出结果:

var age=100;

function test(){

this.age=50;

return function(){

return this.age;

}

}

var m=new test();

alert(m());

var n=test();

alert(n());

**答案：**

1. 50

构造函数一旦返回一个对象，就不再创建新对象

m获得的是function(){ return this.age; }

n=test()，this指向window。先将全局变量age变为50,又返回一个函数function(){ return this.age; }保存在变量n中

调用n时，this指向window。

### 2. 判断以下程序的输出结果:

var name="The Window";

var obj={

name:"My obj",

getName:function(){

return function(){

return this.name;

}

}

};

console.log(obj.getName()());

**答案：**

the window

obj.getName() 返回一个函数对象function(){ return this.name; }

(function(){ return this.name; }()) 相当于匿名函数自调，this指向window

### 3. 判断以下程序的输出结果:

var length=10;

function fn(){

console.log(this.length);

}

var obj={

length:5,

method:function(fn){

fn();

arguments[0]();

}

};

obj.method(fn,1)

**答案：**

10 2

fn() this指向window，所以输出10

arguments[0]() 属于特殊情况，this->arguments,相当于arguments.0(), 所以，this指向arguments。所以length输出的是obj.method()的参数个数，为2.

### 4. 统计一个字符串中出现次数最多的字符是? 共出现多少次

**答案：**

var dict={};

var c="", max=1;

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

var char=str[i];

if(dict[char]===undefined)

dict[char]=1;

else{

dict[char]+=1;

if(dict[char]>max){

max=dict[char];

c=char;

}

}

}

console.log(c,max);

提前创建一个空对象，用于保存每个字母出现的次数。

提前创建变量，准备保存出现次数最多的字符和出现的次数。

然后，遍历字符串中每个字母，每遍历一个字母就判断结果对象中是否包含以当前字母为属性名的属性。如果不包含以当前字母为属性名的属性，说明是首次遇见该字母，就向结果对象中强行添加以该字母为属性名的属性，值暂时为1。如果结果对象中已经包含以当前字母为属性名的属性，说明不是第一次碰见该字母。则取出该字母名属性对应的次数+1。只要当前字母出现的次数>之前变量中记录的最大次数，就用当前字母和出现次数，取而代之。

### 5. 判断以下程序的输出结果:

for(var i=0;i<5;i++){

setTimeout(function(){

console.log(i);

},0)

}

console.log(i);

**答案：**

5 5 5 5 5

函数定义时，函数内容是不执行的，所以i还是i，不会变成0，1，2，3，4

定时器中的回调函数只能再主程序执行完才能开始执行

当主程序执行完，循环变量i，已经被改为5了。

### 6. 判断以下程序的输出结果:

window.color="red";

let color="green";

let obj={

color:"blue"

};

let sayColor=()=>{

return this.color;

}

console.log(sayColor.apply(obj));

let a=10;

console.log(window.a);

**答案：**

red undefined

let相当于匿名函数自调，所以，let声明的变量，不会自动加入到window

剪头函数内外this通用，所以apply也无法替换sayColor函数内的this，所以this指向window，所以输出red

### 7. 判断以下程序的输出结果:

var c=1;

function c(c){

console.log(c);

var c=3;

}

c(2);

**答案：**

报错: TypeError: c不是一个函数

function c(c){} 整体被声明提前，后又被c=1代替。所以，c最后不是一个函数，而是数字1

### 8. 判断以下程序的输出结果:

function change(){

alert(typeof fn)

function fn(){ alert('hello') }

var fn;

}

change();

**答案：**

function

function fn(){…}被整体声明提前了

var fn发现已经有fn变量了，就不再重复创建，所以，var fn没作用。

### 9. 判断以下程序的输出结果:

a=3

a.prop=4;

alert(a+a.prop)

**答案：**

NaN

a.prop=4，等效于new Number(a).prop=4, 但是new Number(a)，使用后自动释放，4也不存在了

再次使用a.prop，又等效于新的new Number(a)，所以没有prop属性，值为undefined。

数字+undefined, undefined隐式转为数字NaN，导致计算结果为NaN

### 10. 判断以下程序的输出结果:

var o={

a:10,

b:{

a:12,

fn:function(){

var a=13;

console.log(this.a);

}

}

}

o.b.fn();

**答案：**

12

this指.前的o.b对象，所以a为12

### 11. 判断以下程序的输出结果:

var obj1 = {

name: 'obj1',

fn: function() {

document.write(this.name);

}

};

var obj2 = {name: 'obj2'};

var obj3 = {name: 'obj3'};

obj1.fn();

var newFn = obj1.fn;

newFn();

newFn.call(obj2);

obj3.fn = newFn;

obj3.fn();

**答案：**

obj1 空字符串 obj2 obj3

this指.前的obj1

因为newFn调用时，前边没有.，所以this->window，

call是强行替换newFn中的this为obj2

this指.前的obj3

### 12. 一个数组 par 中存放有多个人员的信息，每个人员的信息由年龄 age 和姓名 name 组成，如{age: 2, name: 'xx'}。请写一段 JS 程序，对这个数组按年龄从小到大进行排序。

**答案:**

function parSort(arr, propName) {

arr.sort(function(a, b) {

return a[propName]-b[propName];

});

}

parSort(arr, “age”)

1. 数组的sort函数的参数，是一个比较器函数。比较器函数的形参a和b，指当前要排序的数组中的任意两个作比较的元素。如果作比较的a和b两个元素是简单的数字类型，则a直接和b相减，就可比较两数大小。但是，如果a和b都是对象类型的元素。要比较两个对象中某个属性的值得大小，就必须用a.属性-b.属性。如果属性名是灵活的，来自于变量，则必须用a[属性名变量]-b[属性名变量]，就可比出两个对象的某个属性值得大小。
2. 最后，数组是引用类型的对象，在函数内修改数组，等效于修改原数组。所以不用返回值。

### 13. 有字符串 var = 'abc345efgabcab'，请写出 3 条 JS 语句分别实现如下 3 个功能：

1）去掉字符串中的a、b、c 字符，形成结果：'345efg'

2）将字符串中的数字用中括号括起来， 形成结果：'abc[345]efgabcab'

3）将字符串中的每个数字的值分别乘以 2，形成结果：'abc6810efgabcab'

**答案:**

1)str.replace(/([a-c])/g, '');

2)str.replace(/(\d+)/g, '[$1]');

3)str.replace(/(\d)/g, function(num) {return num\*2;});

1. 将字符串中的a，b，c三个字符都替换为空字符串
2. 找到字符串中多个连续的数字，分为一组，然后将这一组的内容，替换为用[]包裹。$1，可获得关键词中第一个()包裹的内容。
3. 找到字符串中每个数字，\*2后，再放回原位置。

### 14. 判断以下程序的输出结果:

var a=10;

var obj={

a:20,

intr:function(){

var a=30;

console.log(this.a);

}

}

obj.intr();

var intr=obj.intr;

intr();

**答案:**

20 10

obj.intr()，this指.前的obj，所以输出20

intr(), this指window，所以输出10

### 15. 判断以下程序的输出结果:

function fun(){

for(var i=0,arr=[];i<3;i++){

arr[i]=function(){

console.log(i);

}

}

return arr;

}

var funs=fun();

funs[0]();

funs[1]();

funs[2]();

**答案:**

3 3 3

### 16. 定义函数实现深克隆一个对象:

**答案:**

var lilei={

sname:"Li Lei",

score:null,

friends:["jack","rose"],

address:{

prov:"北京",

city:"北京",

area:"海淀",

street:"万寿路"

},

sage:11

}

function clone(obj){

if(obj===null){

return null;

}else if({}.toString.call(obj)==="[object Array]"){

var newArr=[];

newArr=obj.slice();

return newArr;

}

var newObj={};

//遍历原obj的每个属性

for(var key in obj){

//如果原对象中当前属性值是原始类型

if(typeof obj[key]!=="object"){

//在新对象中添加和原对象中同名的属性

newObj[key]=obj[key];

//原始类型复制，就是复制副本

}else{//否则,当前属性不是原始类型的值，再次调用clone函数，继续复制当前属性值

newObj[key]=clone(obj[key])

}

}

return newObj;

}

console.log(lilei);

var lilei2=clone(lilei);

console.log(lilei2);

console.log(lilei==lilei2);//true

lilei2.address.area="朝阳";

console.log(lilei.address);

### 17. 介绍 JavaScript 的原型，原型链？有什么特点？

**答案:**

原型：

- JavaScript 的所有对象中都包含了一个 [proto] 内部属性，这个属性所对应的就是该对象的原型

- JavaScript 的函数对象，除了原型 [proto] 之外，还预置了 prototype 属性

- 当函数对象作为构造函数创建实例时，该 prototype 属性值将被作为实例对象的原型 [proto]。

原型链：

- 当一个对象调用的属性/方法自身不存在时，就会去自己 [proto] 关联的前辈 prototype 对象上去找

- 如果没找到，就会去该 prototype 原型 [proto] 关联的前辈 prototype 去找。依次类推，直到找到属性/方法或 undefined 为止。从而形成了所谓的“原型链”

原型特点：

- JavaScript 对象是通过引用来传递的，当修改原型时，与之相关的对象也会继承这一改变

### 18. 谈谈Javascript 垃圾回收方法

**答案:**

标记清除（mark and sweep）

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

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

引用计数(reference counting)

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

### 19. 说说严格模式的限制

**答案:**

严格模式主要有以下限制：

- 变量必须声明后再使用

- 函数的参数不能有同名属性，否则报错

- 不能使用 with 语句

- 不能对只读属性赋值，否则报错

- 不能使用前缀 0 表示八进制数，否则报错

- 不能删除不可删除的属性，否则报错

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

- eval 不会在它的外层作用域引入变量

- eval 和 arguments 不能被重新赋值

- arguments 不会自动反映函数参数的变化

- 不能使用 arguments.callee

- 不能使用 arguments.caller

- 禁止 this 指向全局对象

- 不能使用 fn.caller 和 fn.arguments 获取函数调用的堆栈

- 增加了保留字（比如 protected、static 和 interface）

### 20. 使用正则表达式验证邮箱格式

**答案:**

var reg = /^(\w)+(\.\w+)\*@(\w)+((\.\w{2,3}){1,3})$/;

var email = "example@qq.com";

console.log(reg.test(email)); // true

### 21. 使用typeof bar ===“object”来确定bar是否是一个对象时有什么潜在的缺陷？这个陷阱如何避免？

**答案:**

尽管typeof bar ===“object”是检查bar是否是对象的可靠方法，但JavaScript中令人惊讶的问题null也被认为是一个对象！

因此，对于大多数开发人员来说，下面的代码会将true（而不是false）打印到控制台：

var bar = null;

console.log(typeof bar === "object"); // logs true!

只要知道这一点，就可以通过检查bar是否为空来轻松避免该问题：

console.log((bar !== null) && (typeof bar === "object")); // logs false

为了让我们的答案更加的完整，还有两件事值得注意： 首先，如果bar是一个函数，上面的解决方案将返回false。在大多数情况下，这是所期望的行为，但是在您希望函数返回true的情况下，您可以将上述解决方案修改为：

console.log((bar !== null) && ((typeof bar === "object") || (typeof bar === "function")));

其次，如果bar是数组，则上述解决方案将返回true（例如，如果var bar = [];）。在大多数情况下，这是所希望的行为，因为数组确实是对象，但是在您想要对数组也是false的情况下，可以将上述解决方案修改为：

console.log((bar !== null) && (typeof bar === "object") && (toString.call(bar) !== "[object Array]"));

但是，还有一个替代方法对空值，数组和函数返回false，但对于对象则为true：

console.log((bar !== null) && (bar.constructor === Object));

或者，如果您使用jQuery：

console.log((bar !== null) && (typeof bar === "object") && (! $.isArray(bar)));

ES5使得数组的情况非常简单，包括它自己的空检查：

console.log(Array.isArray(bar));

### 22. 以下代码的输出是什么？解释你的答案

var a={},

b={key:'b'},

c={key:'c'};

a[b]=123;

a[c]=456;

console.log(a[b]);

**答案:**

456

原因如下：设置对象属性时，JavaScript会隐式地将参数值串联起来。在这种情况下，由于b和c都是对象，它们都将被转换为“[object Object]”。因此，a [b]和a [c]都等价于[“[object Object]”]，并且可以互换使用。因此，设置或引用[c]与设置或引用[b]完全相同。

## 二、DOM

### 1. 利用冒泡和不利用冒泡的差别

**答案：**

1. 绑定位置不同: 不利用冒泡绑定在目标元素上，利用冒泡绑定在父元素上
2. 监听对象的个数不同: 不利用冒泡会反复创建多个监听，利用冒泡始终只有一个监听
3. 动态生成的元素: 不利用冒泡无法自动获得事件处理函数，必须反复绑定

利用冒泡可让动态添加的子元素自动获得父元素的处理函数，无需反复绑定

### 2. 按HTML查找和按选择器查找的差别

**答案：**

1. 返回值不同: 按HTML查找返回动态集合，按选择器查找返回非动态集合
2. 效率不同: 按HTML查找效率高，按选择器查找效率低
3. 易用性不同: 当条件复杂时，按HTML查找繁琐，而按选择器查找简单

### 3. 列举DOM中常用优化

**答案：**

1. 查找时，如果之用一个条件就可查询出结果时，优先选择按HTML查找。如果查找条件复杂，则优先选择易用的按选择器查找
2. 添加时，尽量减少操作DOM树的次数，减少重排重绘。如果同时添加父元素和子元素，应先将子元素添加到到父元素，最后再将父元素添加到DOM树。如果添加多个平级子元素，则应先将子元素添加到文档片段，最后，再将文档片段添加到DOM树
3. 修改时，尽量减少重排重绘。如果同时修改多个元素的内容或样式，应尽量使用innerHTML和cssText方式修改元素的内容和样式。应使用class批量修改样式
4. 事件绑定时，应尽量利用冒泡减少事件监听的个数。

### 4. 如何鉴别浏览器的名称和版本号

**答案：**

var browser,version,ua=navigator.userAgent;

if(ua.indexOf("IE")!=-1) browser="IE"

else if(ua.indexOf("Edge")!=-1) browser="Edge"

else if(ua.indexOf("Firefox")!=-1) browser="Firefox"

else if(ua.indexOf("OPR")!=-1) browser="OPR"

else if(ua.indexOf("Chrome")!=-1) browser="Chrome"

else if(ua.indexOf("Safari")!=-1) browser="Safari"

else if(ua.indexOf("Trident")!=-1) browser="IE",version=11;

document.write(`<h1>${browser}</h1>`);

if(version===undefined){

//截取: 从浏览器名称所在位置，再跳过浏览器名称长度+1 之后的3位

var i=ua.indexOf(browser);

i+=browser.length+1;

version=parseFloat(ua.slice(i,i+3))

}

document.write(`<h1>${version}</h1>`);

### 5. 为按钮绑定事件，实现事件节流:

**答案:**

//节流

var canClick=true;

btn.onclick=function(){

if(canClick){

canClick=false;

console.log("发送ajax请求，加载更多...");

setTimeout(function(){

console.log("请求完成!");

canClick=true;

},3000);

}

}

### 6. 为页面绑定滚动事件，实现事件防抖:

**答案:**

//防抖

var timer1;

window.onscroll=function(){

if(timer1!==undefined){

clearTimeout(timer1);

}

timer1=setTimeout(function(){

console.log("发送ajax请求，加载更多");

timer1=undefined;

},200)

}

### 7. DOM 元素 e 的 e.getAttribute(propName)和 e.propName 有什么区别和联系

**答案:**

- e.getAttribute()，是标准 DOM 操作文档元素属性的方法，具有通用性可在任意文档上使用，返回元素在源文件中设置的属性

- e.propName 通常是在 HTML 文档中访问特定元素的特性，浏览器解析元素后生成对应对象（如 a 标签生成 HTMLAnchorElement），这些对象的特性会根据特定规则结合属性设置得到，对于没有对应特性的属性，只能使用 getAttribute 进行访问

- e.getAttribute()返回值是源文件中设置的值，类型是字符串或者 null（有的实现返回""）

- e.propName 返回值可能是字符串、布尔值、对象、undefined 等

- 大部分 attribute 与 property 是一一对应关系，修改其中一个会影响另一个，如 id，title 等属性

- 一些布尔属性`<input hidden/>`的检测设置需要 hasAttribute 和 removeAttribute 来完成，或者设置对应 property

- 像`<a href="../index.html">link</a>`中 href 属性，转换成 property 的时候需要通过转换得到完整 URL

- 一些 attribute 和 property 不是一一对应如：form 控件中`<input value="hello"/>`对应的是 defaultValue，修改或设置 value property 修改的是控件当前值，setAttribute 修改 value 属性不会改变 value property

### 8. 如何最小化重绘(repaint)和回流(reflow)？

**答案:**

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

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

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

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

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

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

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

### 9. 描述一下DOM事件模型：

**答案:**

捕获、目标触发、冒泡

捕获阶段: 由外向内，依次记录各级父元素上的事件处理函数。只记录，暂不触发。

目标触发：优先触发目标元素上的事件处理函数。

冒泡: 由内向外，依次触发各级父元素上的事件处理函数。

### 10. 谈谈事件委托/代理:

**答案:**

事件委托是指将事件绑定目标元素的到父元素上，利用冒泡机制触发该事件

优点：

- 可以减少事件注册，节省大量内存占用

- 可以将事件应用于动态添加的子元素上

缺点： 使用不当会造成事件在不应该触发时触发

示例：

js

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

var target = event.target || event.srcElement;

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

console.log(target.innerHTML);

}

}, false);

### 11. IE 的事件处理和 W3C 的事件处理有哪些区别？

**答案:**

绑定事件

- W3C: targetEl.addEventListener('click', handler, false);

- IE: targetEl.attachEvent('onclick', handler);

删除事件

- W3C: targetEl.removeEventListener('click', handler, false);

- IE: targetEl.detachEvent(event, handler);

事件对象

- W3C: var e = arguments.callee.caller.arguments[0]

- IE: window.event

事件目标

- W3C: e.target

- IE: window.event.srcElement

阻止事件默认行为

- W3C: e.preventDefault()

- IE: window.event.returnValue = false'

阻止事件传播

- W3C: e.stopPropagation()

- IE: window.event.cancelBubble = true

### 12. 解释一下这段代码的意思吗？

```js

[].forEach.call($$("\*"), function(el){

el.style.outline = "1px solid #" + (~~(Math.random()\*(1<<24))).toString(16);

})

```

**答案:**

获取页面所有的元素，遍历这些元素，为它们添加 1 像素随机颜色的轮廓(outline)

- $$(sel) // $$函数被许多现代浏览器命令行支持，等价于 document.querySelectorAll(sel)

- [].forEach.call(NodeLists) // 使用 call 函数将数组遍历函数 forEach 应到节点元素列表

- el.style.outline = "1px solid #333" // 样式 outline 位于盒模型之外，不影响元素布局位置

- (1<<24) // parseInt("ffffff", 16) == 16777215 == 2^24 - 1 // 1<<24 == 2^24 == 16777216

- Math.random()\\*(1<<24) // 表示一个位于 0 到 16777216 之间的随机浮点数

- ~~Math.random()\\*(1<<24) // ~~ 作用相当于 parseInt 取整

- (~~(Math.random()\\*(1<<24))).toString(16) // 转换为一个十六进制-

### 13. 前端优化的方法：

**答案:**

1.减少dom操作

    2.部署前，图片压缩，代码压缩

    3.优化js代码结构，减少冗余代码

    4.减少http请求，合理设置 HTTP缓存

    5.使用内容分发cdn加速

    6.静态资源缓存

    7.图片延迟加载

## 三、jQuery

### 1. $的原理

**答案：**

1. $(“选择器”) 是先查找DOM元素，再将DOM元素放入jQuery对象中

其中自带优化:

如果选择器是#id，则自动调用getElementById

如果选择器是.class，则自动调用getElementsByClassName

如果选择器是标签名，则自动调用getElementsByTagName

否则，其它选择器，都自动调用querySelectorAll()

1. $(DOM元素) 是直接将DOM元素放入jQuery对象中
2. $(“HTML片段”) 是创建一个新元素
3. $(function(){}) 是绑定事件，在DOM内容加载后就提前触发。

### 2. 实现动画有几种方式，哪种好？

**答案：**

1. CSS: transition, animateion

优点: 由专门的排版引擎解析，效率高

缺点: 无法随意控制交互行为

1. JS: 定时器， $().animate()

优点: 可随意控制交互行为

缺点: 效率不如css动画

1. requestAnimationFrame()

优点: 可根据浏览器的刷新频率自动优化动画效果

缺点: 新API，有兼容性问题

### 3. 实现跨域访问有几种方式

**答案：**

主要有两种

1. JSONP:

客户端: 客户端动态添加script元素，用script发送请求，代替ajax请求，并携带客户端一个函数名到服务端

服务端: 接收客户端发来的函数名，将函数名和要返回的数据拼接为一条可执行的js语句，返回

1. CORS: Cross-Origin Resources Sharing

客户端正常发送ajax请求，服务端定义响应头，允许指定来源的请求跨域访问:

res.writeHead(200,{

…，

“Access-Control-Allow-Origin”:”允许的请求来源域名”

})

### 4. 请解释一下 JavaScript 的同源策略, 为什么要有同源限制

**答案：**

概念:同源策略是客户端脚本（尤其是 Javascript）的重要的安全度量标准。它最早出自 Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议

- 指一段脚本只能读取来自同一来源的窗口和文档的属性

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

缺点: 现在网站的 JS 都会进行压缩，一些文件用了严格模式，而另一些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节

### 5. $(document).ready() 是个什么函数？为什么要用它？

**答案：**

这个问题很重要，并且常常被问到。 ready() 函数用于在文档进入ready状态时执行代码。当DOM 完全加载（例如HTML被完全解析DOM树构建完成时），jQuery允许你执行代码。使用$(document).ready()的最大好处在于它适用于所有浏览器，jQuery帮你解决了跨浏览器的难题。

### 6. JavaScript window.onload 事件和 jQuery ready 函数有何不同？

**答案：**

这个问答是紧接着上一个的。JavaScript window.onload 事件和 jQuery ready 函数之间的主要区别是，前者除了要等待 DOM 被创建还要等到包括大型图片、音频、视频在内的所有外部资源都完全加载。如果加载图片和媒体内容花费了大量时间，用户就会感受到定义在 window.onload 事件上的代码在执行时有明显的延迟。

另一方面，jQuery ready() 函数只需对 DOM 树的等待，而无需对图像或外部资源加载的等待，从而执行起来更快。使用 jQuery $(document).ready() 的另一个优势是你可以在网页里多次使用它，浏览器会按它们在 HTML 页面里出现的顺序执行它们，相反对于 onload 技术而言，只能在单一函数里使用。鉴于这个好处，用 jQuery ready() 函数比用 JavaScript window.onload 事件要更好些。

## 四、Vue

### 1. Vue的双向数据绑定原理是什么？

**答案：**

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

具体步骤：

第一步：需要observe的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter和getter

这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

第二步：compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

第三步：Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是:

1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个update()方法

3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

### 2. 请详细说下你对vue生命周期的理解

**答案：**

总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： 在beforeCreated阶段，vue实例的挂载元素$el和数据对象data都为undefined，还未初始化。在created阶段，vue实例的数据对象data有了，$el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

### 3. 封装 vue 组件的过程

**答案：**

首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：效率低、难维护、复用性等问题。

然后，使用Vue.extend方法创建一个组件，然后使用Vue.component方法注册组件。子组件需要数据，可以在props中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用emit方法。

### 4. mvc与mvvm

**答案：**

MVC

    模型－视图－控制器（Model-View-Controller）

    Model和View永远不能相互通信，只能通过Controller传递。

    Controller可以直接与Model对话（读写调用Model），Model通过Notification和KVO机制与Controller间接通信。

    Controller可以直接与View对话，通过outlet,直接操作View,outlet直接对应到View中的控件,View通过action向Controller报告事件的发生(如用户Touch我了)。Controller是View的直接数据源（数据很可能是Controller从Model中取得并经过加工了）。Controller是View的代理（delegate),以同步View与Controller。

    MVVM

    Model －ViewModel － View

    什么是 MVVM：一个 MVC 的增强版，我们正式连接了视图和控制器，并将表示逻辑从 Controller 移出放到一个新的对象里，即 View Model。MVVM 听起来很复杂，但它本质上就是一个精心优化的 MVC 架构

    Model层是少不了的了，我们得有东西充当DTO(数据传输对象)，当然，用字典也是可以的，编程么，要灵活一些。Model层是比较薄的一层，如果学过Java的小伙伴的话，对JavaBean应该不陌生吧。

    ViewModel层，就是View和Model层的粘合剂，他是一个放置用户输入验证逻辑，视图显示逻辑，发起网络请求和其他各种各样的代码的极好的地方。说白了，就是把原来ViewController层的业务逻辑和页面逻辑等剥离出来放到ViewModel层。

    View层，就是ViewController层，他的任务就是从ViewModel层获取数据，然后显示。

### 5. Vue首屏加载非常慢，如何解决?

**答案：**

Vue首屏加载非常慢

原因: 当打包应用时，将所有JavaScript代码打包在一个文件中，导致js代码非常庞大，严重影响页面加载速度。

解决:

1. 配置打包工具，将组件分别打包到不同的js代码块中

build/webpack.base.conf.js

output:{

path: config.build.assetsRoot,

filename:’[name].js’,

//新增

chunkFilename:”[name].js”,

publicPath: process.env.NODE\_ENV===”production”

?config.build.assetsPublicPath

:config.dev.assetsPublicPath

}

2. 当路由请求到该组件时，才动态加载组件的内容

路由字典中，路由配置和以前完全一样

但是在引入组件对象时:

import Index from ‘@/views/Index.vue’

改为

const Index=()=>import(‘@/views/Index.vue’)//仅定义函数

//暂未执行

//暂时不引入Index.vue

当用户在vue中请求当前组件对应的路由地址时，由vue-router自动调用加载函数，动态请求Index.vue组件对象

### 6. 实现订阅/发布者模式？

**答案：**

var ublisher = {};

// 定义发布者

publisher.list =[];

// 缓存列表 存放订阅者回调函数

// 增加订阅者

publisher.listen =function(fn){

publisher.list.push(fn);

// 订阅消息添加到缓存列表

}

// 发布消息

publisher.trigger =function(){

for(vari = 0,fn; fn = this.list[i++];){

var that =this

fn.apply(null,arguments);

}

}

### 7. 什么是虚拟DOM树:

**答案：**

什么是: 仅包含可能变化的节点和可能变化的属性的树结构

<body>

<div id=”app”>

<img src=”logo.png”> alt title id ….

<h1>{{uname}}</h1>id class title name

<hr>

<h2>{{score}}</h2>

</div>

{

el:”#app”,

children:[

{el:”h1”, innerText:uname},

{el:”h2”,innerText:score}

]

}

为什么: 内容少，便于快速遍历比较不同

如何发挥作用:

当data中模型变量改变时，会通知虚拟DOM树

虚拟DOM树先缓存本次的修改在元素对象上

将一批修改生成新的DOM子树和原虚拟DOM树做对比。

一旦发现不同的元素或内容，就只更新有修改的元素

虚拟DOM树中，封装了传统DOM API: createElement() appendChild() .innerHTML，避免了程序员编写大量重复的代码。

### 8. Vue 如何去除url中的 #

**答案：**

vue-router 默认使用 hash 模式，所以在路由加载的时候，项目中的 url 会自带 #。如果不想使用 #， 可以使用 vue-router 的另一种模式 history

new Router({

mode: 'history',

routes: [ ]

})

需要注意的是，当我们启用 history 模式的时候，由于我们的项目是一个单页面应用，所以在路由跳转的时候，就会出现访问不到静态资源而出现 404 的情况，这时候就需要服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面

### 9. Vue组件间如何通信:

**答案：**

组件通信

父组件向子组件通信

子组件通过 props 属性，绑定父组件数据，实现双方通信

子组件向父组件通信

将父组件的事件在子组件中通过 $emit 触发

非父子组件、兄弟组件之间的数据传递

/\*新建一个Vue实例作为中央事件总嫌\*/

let event = new Vue();

/\*监听事件\*/

event.$on('eventName', (val) => {

//......do something

});

/\*触发事件\*/

event.$emit('eventName', 'this is a message.')