

1、写一段脚本，实现观察者模式

2、js 输出当前访问页面的设备的类型

```
function checkEquipment() {
    var output = {};
    if (navigator.userAgent.match(/(iPhone|iPod|iPad);?/i)) {
        output['ios'] = true;
    } else if (
        navigator.userAgent.match(/android/i)) {
        output['android'] = true;
    } else if (navigator.userAgent.match(/windows/i)) {
        output['windows'] = true;
    } return output;
};
console.log(checkEquipment())//对应输出其中一个: {android: true} {ios: true}
{windows: true}
```

通过 navigator 属性检查浏览器是什么浏览器 navigator.userAgent;

3、写一段脚本，以你认为最优的方式向<body>中添加 10 个<p>Hello World</p>然后将所有的 p 标签删除，

4、以您认为合理的方式，给下面 HTML 元素绑定 click 事件, 使绑定事件之后添加的 li 依然可以触发事修的处理函数

```
<ul><li>1</li><li>2</li><li>3</li><li>4</li><li>5</li></ul>
```

错误方式: **55555**

```
var lis = document.getElementsByTagName('li')
var len = lis.length;
for (var i = 0; i < len; i++) {
    $(lis[i]).click(function () {
        alert(i);
        //555 })
    })
}
```

正确方式一(闭包)**01234**

```
var lis = document.getElementsByTagName('li');
for (var i = 0; i < lis.length; i++) {
    (function (index) {
        lis [i].onclick = function () {
            alert(index);
        }
    })(i);
};
```

正确方式二(利用 jq 中的 each) **01234**

```
var lis = $("ul>li");
```

```
$.each(lis, function (index, element) {
    $(element).click(function () {
        alert(index); //索引
    })
})
```

## 5、编写一段脚本，去除以下数组中重复元素

```
var arr = [1, 2, 3, '3', 4, 2]
console.log(arr)
```

方法一:

```
Array.prototype.unique = function () {
    var res = [];
    for (var i = 0, newArr = []; i < this.length; i++) {
        if (!newArr[this[i]]) {
            res.push(arr[i]);
            newArr[this[i]] = 1;
        }
    }
    return res;
}
var arr = [1, 2, 3, '3', 4, 2];
console.log(arr.unique()); // [1, 2, 3, 4]
```

方法二:

```
let arr = [1, 2, 3, '3', 4, 2]
let set = new Set(arr);
let newArr = Array.from(set);
console.log(newArr); // [1, 2, 3, "3", 4]
```

方法三:

```
Array.prototype.unique=function(){
    var newArr=[];
    for(var i=0;i<this.length;i++){
        if(newArr.indexOf(this[i])==-1){
            newArr.push(this[i]);
        }
    }
    return newArr;
}
var arr= [1,2,3,'3','3',2,3,4,2];
console.log(arr.unique()); // [1, 2, 3, "3", 4]
```

## 6、请问下述代码最终结果是什么？ 打印结果：55555

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

```

setTimeout(function(){
    console.log(i)//55555
},1000)
//console.log('i',i)//01234 这里能正常打印
}

```

原因分析: `setTimeout()` 是一个异步处理函数, 它会等待所有的主线程任务处理完, 才开始执行自己的内部的任务, 每隔 1s 往任务队列中添加一个任务【闭包函数, `setTimeout()` 中的函数, 现在还没执行】, 当主线程执行完时, 这时 `i=5`,

才开始执行任务队列中的任务【闭包函数, `setTimeout()` 中的函数开始执行, 执行 5 次】。

`for` 循环括号内的就是主线程, 执行完时 `i` 是 5, 所以会打印出 5 次 5;

如果想打印出 0, 1, 2, 3, 4

解决方案:

1. 把 `var` 改为 `let`, `let` 是块级作用域, 每次 `for` 循环都会把对应的 `i` 绑定到添加的任务【闭包函数, `setTimeout()` 中的函数】中, 所以当主线程执行完时, 也不会影响到每个任务中 `i`。所以可以打印出 0 1 2 3 4
2. 把定时器放在一个自执行函数中用 `i` 当做参数

```

for(var i=0;i<5;i++){
    (function(i){
        setTimeout(function(){
            console.log(i);//01234
        },1000)
    })(i)
}

```

## 7、填充/TODO: 处代码, 以达成克隆目标对象的目的

```

var sourceObject = {
    id: 0,
    name: 'Alan',
    scores: [1, 2, 3],
    favorite: {
        tag1: 'chinese',
        tag2: 'math',
    }
}

```

```

var destObject = clone(sourceObject);

```

```

function clone(obj) {
    var type = typeof (obj);
    if (type === "string" || type === "boolean" || type === "number") {
        var newone = obj;
    } else if (type === "object") {
        if (obj === null) {
            var newone = null;
        } else if (Object.prototype.toString.call(obj).slice(8, -1) === "Array") {
            var newone = [];

```

```

        for (var i = 0; i < obj.length; i++) {
            newone.push(clone(obj[i]));
        }
    } else if (Object.prototype.toString.call(obj).slice(8, -1) === "Object") {
        var newone = {};
        for (var i in obj) {
            newone[i] = clone(obj[i]);
        }
    }
}
return newone;
}

```

this.constructor 是什么?

### constructor:

- 1、constructor 始终指向创建当前对象的构造（初始化）函数。
- 2、每个函数都有一个默认的属性 prototype，而这个 prototype 的 constructor 默认指向这个函数

obj 改成数组 var obj = [1, 2]->this.constructor 值就是  
 f Array() { [native code] } this.constructor===Array 成立

obj 改成对象 var obj = { a: 1 }->this.constructor 值就是  
 f Object() { [native code] } this.constructor===Object 成立

```

Object.prototype.clone = function () {
    console.log(this.constructor)//f Object() { [native code] }
}
var obj = { a: 1 }
var obj2 = obj.clone()

```

其他原型复制方法通用

```

Object.prototype.clone = function () {
    var o = (this.constructor === Array ? [] : {});//{}
    for (var key in this) { //this -> {name: "johnny"}
        o[key]=typeof this[key]=== "object"?this[key].clone() : this[key];
    }
    return o;
}
var destObject = sourceObject.clone();//使用

```

### 7、编写一段脚本，实现 Function.bind 函数相同的功能

```

Function.prototype.bind = function (context) {
    var self = this; // 保存原函数
    return function () { // 返回一个新函数

```

```

    return self.apply(context, arguments); // 执行新函数时，将传入的上下文 context 作为
新函数的 this
  }
}

```

8、请用纯编写一个下拉菜单

10、通过继承的方式达到输出结果 123

```

var AClass = function() {
    this.id=123;
}
var BClass=function(id) {
    AClass.call(this,id)
}
var bInstance = new BClass;
BClass.prototype.sayId=function(){
    console.log(this.id)
}

```

bInstance.sayId();

11、为 div 设置类 a 与 b，应编写 HTML 代码? <div class="a b"> </div>

12、设置 CSS 属性 clear 的值为( none)可清除左右两边浮动

13、css 属性(margin)可为元素设置外补丁

14、设置文字不换行，超出部分用... 代替样式为：

```
overflow: hidden; text-overflow: ellipsis; white-space: nowrap; width: 100px;
```

15、定义内联（非块状）元素可以定义宽度和高度？

block:将行级别元素转为块级别元素 不换行

附：

inline-block:不能自动换行,但是可以设置宽高

inline:将块级别元素转为行级别元素

16、如何使得英文单词不发生词内断行？

word-break: break-word;

17、写出定义标题的方法，br 标签在 XHTML 中语义是？ 换行

18、不换行必须设置哪些？ white-space: nowrap;

19、在使用 table 表现数据时，有时候表现出来的会比自己实际设置的宽度要宽，为些需要设置哪些属性值？

cellpadding=" 0" , cellspacing=" 0"

20、display:none 与 visibility:hidden 的区别是什么？

1.display:none 是彻底消失，不在文档流中占位，浏览器也不会解析该元素；visibility:hidden 是视觉上消失了，可以理解为透明度为 0 的效果，在文档流中占位，浏览器会解析该元素；

2.使用 visibility:hidden 比 display:none 性能上要好，display:none 切换显示时 visibility，页面产生回流（当页面中的一部分元素需要改变规模尺寸、布局、显示隐藏等，页面重新构建，此时就是回流。所有页面第一次加载时需要产生一次回流），而 visibility 切换是否显示时则不会引起回流。

21: 结果? f a() {}

```

(function (a) {
    console.log(a); //f a(){}
    var a = 10;
}

```

```
function a() { }  
})(100)
```

解释： a 可以声明提前，赋值不提前，函数可以随时调取，这里 a 不是参数是变量了

函数在预解析的时候会被提到最前面，这个函数名和变量名重名，会显示函数

把 function a() {} 改成 function b() {} 就能显示 100 了

100 没起作用是因为局部变量优先规则

```
(function (a) {  
    var a;  
    console.log(a)  
    a = 10;  
    function a() { }  
})(100)
```

局部变量优先原则，原理同下：

```
var a = 5;  
function fn() {  
    var a = 10;  
    console.log(a)    // 10，局部变量优先，在局部找到 a 后，不会再向外查找  
}
```

结果：undefined

```
var a = 10;  
(function () {  
    console.log(a); // undefined  
    var a = 100;  
})();
```

解释：

声明会提升到 function 最开头，但赋值发生在最后上面的代码等价于：打印 a 的时候，a 并没有在 function 内赋值，所以是 undefined

```
(function () {  
    var a;  
    console.log(a);  
    a = 100;  
})();
```

22、编写一个脚本，0 到 10 完成 10 随机整数的排序

```
function sortNumber(a, b) {  
    return a - b; // 升序  
    // return b - a; // 降序  
}  
var iArray = [];  
function getRandom(iStart, iEnd) {
```

```

    var iChoice = iStart - iEnd;
    return Math.abs(Math.ceil(Math.random() * iChoice)) + iStart;
//ceil() 方法可对一个数进行上舍入。
}
for (var i = 0; i < 10; i++) {
    iArray.push(getRandom(1, 10))
}
iArray.sort(sortNumber);
console.log(iArray)//随机生成 [1, 1, 4, 5, 6, 6, 6, 6, 7, 9]

```

23、写出 ul ol dl 三种列表的 html 结构

前两个都是 li

```
<dl><dt>标题</dt><dd>内容 1</dd></dl>
```

24、css 代码优化

```

margin-top:20px;margin-right:5px;margin-left:5px;margin-bottom:20px;font-style:italic;
font-weight:bold;font-size:1em;line-height:140%;font-family:'lucida Grande', sans-serif;
color:#333333;

```

参考：

```

margin: 20px5px;
font: italicssmall-capsbold1em/140%'Lucida Grande', sans-serif;
color: #369;

```

25、针对不同浏览器写出 box-shadow 的格式

```
-webkit-box-shadow: 3px 3px 3px;
```

```
-moz-box-shadow: 3px 3px 3px;
```

```
box-shadow: 3px 3px 3px;
```

26、HTML 有哪些新的页面元素？列给 5 个以上；

Article、header footer figure（一组媒体及标签文字） mark video section embed 外部插件或对象

27、你认为良好的 CSS 架构目标是什么？

28、AJAX 是什么，有什么优点和缺点？

优点：

页面局部刷新，提高用户体验度；

使用异步方式与服务器通信，具有更加迅速的响应能力；

减轻服务器负担；

基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

缺点：不支持浏览器 back 按钮；安全问题；对搜索引擎的支持比较弱。

**AJAX 干掉了 Back 和 History 功能，即对浏览器机制的破坏。**无法回到前一个页面状态

Ajax 就如同**对企业数据建立了一个直接通道**。使开发者不经意间暴露比以前更多的数据和服务器逻辑

对搜索引擎的支持比较弱。如果使用不当，AJAX 会增大网络数据的流量，从而降低整个系统的性能。

29、模块加载器有何优点，以及常模块加载器有哪些

require 加载模块。

module.exports 对外暴露接口。

一个文件就是一个模块，具备独立的作用域，不会污染全局。

30、介绍你所知道的 CSS hack 技巧

```
.bb{
  background-color:#f1ee18; /*所有识别*/
  .background-color:#00deff\9; /*IE6、7、8 识别*/
+background-color:#a200ff; /*IE6、7 识别*/
  _background-color:#1e0bd1; /*IE6 识别*/
}
```

### 31、列举常用 WEB 的性能优化方法

网页内容：

减少 HTTP 请求次数、缓存 AJAX、延迟加载、减少 DOM 元素数量、减少 iframe 数量、减少 cookie 大小

Css:将样式表置顶将、避免 css 表达式、用 link 代替@import、避免使用 filters、css 文件合并与压缩

Javascript:

将脚本置底（将脚本内容在页面信息内容加载后再加载）

使用外部 javascript 和 css 文件

去除重复脚本，避免重复的资源请求

减少 DOM 访问（修改和访问 DOM 元素会造成页面的重绘和重排，循环对 DOM 操作更是减慢页面加载速度）

js 文件合并与压缩

图片方面：优化图片压缩、优化 css Sprite

### 32 浏览器中数据存储有哪些方式？分别有什么区别？

之前 Cookie 实现，localStorage：适用于长期存储数据，浏览器关闭后数据不丢失，数据只能由用户删除；

sessionStorage：存储的数据在浏览器关闭后自动删除。

### 33、多个页面之间如果进行通信，组件之间有哪些传值的方法？

1、父组件往子组件传值 props

2、子组件往父组件传值，通过 emit 事件

### 3、vuex 进行传值

vuex 主要是是做数据交互，父子组件传值可以很容易办到，但是兄弟组件间传值（兄弟组件下又有父子组件），或者大型 spa 单页面框架项目，页面多并且一层嵌套一层的传值，异常麻烦，用 vuex 来维护共有的状态或数据会显得得心应手。

4、provide/inject：主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

5、\$attrs/\$listeners

\$attrs：包含了父作用域中不被 prop 所识别（且获取）的特性绑定（class 和 style 除外）。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定（class 和 style 除外），并且可以通过 v-bind="\$attrs" 传入内部组件。通常配合 inheritAttrs 选项一起使用。

\$listeners：包含了父作用域中的（不含 .native 修饰器的）v-on 事件监听器。它可以通过 v-on="\$listeners" 传入内部组件

### 34、为什么会出现跨域的问题以及解决方式？

### 35、写出以下执行结果，谈谈上下文和作用域的区别？

结果：

~undefined



```
this:1
return1:2
return2:3
setTimeout:3
```

```
var a = 1;
function test() {
  console.log('~' + a); //声明提前
  var a = 2;
  console.log('this:' + this.a) //全局
  setTimeout(function () {
    console.log('setTimeout:' + a)
  }, 0)
  return function () {
    console.log('return1:' + a)
    a = 3
    console.log('return2:' + a)
  }
}
test();
```

### 36、写下结果：说说 Event Loop Task Queue? 23541

Promise 定义之后便会立即执行,其次主任务,其后的.then() 然后 setTimeout

Promise 比 setTimeout() 先执行

执行顺序 Promise-> 全局 ->then->setTimeout

```
setTimeout(function () {
  console.log(1)
}, 0)
new Promise(function executor(resolve) {
  console.log(2)
  for (var i = 0; i < 10000; i++) {
    i == 9999 && resolve();
  }
  console.log(3)
}).then(function () {
  console.log(4)
})
console.log(5)
```

### 37、Vue 中给 data 中的对象属性添加一个新的属性时会发生什么，如何解决？

解：点击 button 会发现，obj.b 已经成功添加，但是视图并未刷新：

因为 vue 底层的东西本身就不支持对数组某一个特定元素的监控，push，pop 这些都能监控到。

对 json 或数组的添加和删除是监听不到的

不是 VUE 的 bug，是 VUE 依赖底层库的问题，

vue 依赖机制 observe --数据监听

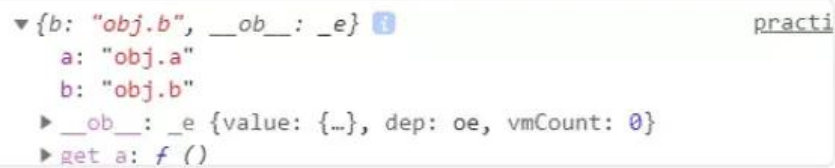
并不是所有的数据操作都能被监听到:

这时就需要使用 Vue 的全局 api—— `$set()`:

vue 实例.`$set(数据, key, val)`

`Vue.set(数据, key, val)` // 直接在 Vue 类上

```
addObjB() {  
  // this.obj.b = 'obj.b' 下面两种写法  
  this.$set(this.obj, 'b', 'obj.b')  
  Vue.set(this.obj, 'b', 'obj.b')  
  console.log(this.obj)  
}
```



```
▼ {b: "obj.b", __ob__: _e} ⓘ  
  a: "obj.a"  
  b: "obj.b"  
  ► __ob__: _e {value: {...}, dep: oe, vmCount: 0}  
  ► get a: f ()
```

• obj.a

添加obj.b

```
<div id='app'>  
  <ul><li v-for="value in obj" :key="value">{{value}}</li> </ul>  
  <button @click="addObjB">添加 obj.b</button>  
</div>  
<script src='vue2.js'></script>  
<script type="text/javascript">  
  var vm = new Vue({  
    el: '#app',  
    data() {  
      obj: {  
        a: 'obj.a'  
      }  
    },  
    methods: {  
      addObjB() {  
        this.obj.b = 'obj.b'  
        console.log(this.obj)  
      }  
    }  
  })  
</script>
```

### 38、Vue 路由的钩子函数有哪些，并列举实际应用

守卫作用：过来新的地址要先经过守卫同意了才能进去 也就是 next()

分类：全局和局部

路由钩子函数有三种：

- 1：全局钩子： beforeEach、afterEach
- 2：单个路由里面的钩子： beforeEnter、beforeLeave
- 3：局部组件路由： beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

### 39、call apply bind 函数存在的区别

相似之处

1. 都是用来改变函数的 this 对象的指向的
2. 第一个参数都是 this 要指向的对象
3. 都可以利用后续参数传参

区别

1. **call 和 apply 都是对函数的直接调用（也叫直接执行函数），而 bind 方法返回的仍然是一个函数**，因此后面还需要 () 来进行调用才可以（将上下文绑定到 bind() 括号中的参数上，然后将它返回）。所以，bind 后函数不会执行，而只是**返回一个改变了上下文的函数副本**。
2. call 和 apply 都可以传参数。call 后面的参数与 fn 方法中是一一对应的，而 **apply 的第二个参数是一个数组**，数组中的元素是和 fn 方法中一一对应的，这就是两者最大的区别。
3. bind 是 ES5 中的方法，可以向 call 一样传参，也可以在调用的时候再进行传参。

总结

1. 当我们使用一个函数需要改变 this 指向的时候才会用到 call apply bind
2. 如果你要传递的参数不多，则可以使用 fn.call(thisObj, arg1, arg2 ...)
3. 如果你要传递的参数很多，则可以用数组将参数整理好调用 fn.apply(thisObj, [arg1, arg2 ...])
4. 如果你想**生成一个新的函数长期绑定某个函数给某个对象使用**，则可以使用 const newFn = fn.bind(thisObj); newFn(arg1, arg2...)

### 40、判断真假：

结果 false false true false

因为：typeof null ->Object

typeof NaN ->number

null instanceof object null === undefined null==undefined NaN==NaN

### 41、foo 对象有 att 属性，那么获取 att 属性的值，以下哪些做法对

getAttribute() 和 attr() 都是获取元素属性的方法前者 js 写法后者 jquery 写法

这里不能用

正确写法：

foo.att、foo['att']

错误写法：

foo('att')、foo{'att'}、foo['a','t','t']、foo.getAttribute('att')、foo.attr('att')

### 42、以下哪个方法用于 Array 中删除元素，并向数组添加新元素—>splice

splice push shift slice

#### 43、解释一下为什么需要清除浮动？清除浮动的方式

浮动导致元素已不在普通流中，所以在排列布局的时候文档中的普通流表现的和浮动元素不存在一样，当浮动元素的高度超出包含框的时候，会出现包含框不会自动撑高来包裹浮动元素，即所谓的“高度塌陷”。  
换种说法：父元素的高度是由子元素撑开的，且子元素设置了浮动，父元素没有设置浮动，子元素脱离了标准的文档流，那么父元素的高度会将其忽略，如果不清除浮动，父元素会出现高度不够，那样如果设置 border 或者 background 都得不到正确的解析

**清除浮动:**在父元素的最后加一个冗余元素并为其设置 clear:both

给父元素添加 overflow:hidden ||auto

#### 44、编写程序，请用 javascript 编写获取 url 中的参数值, 要求至少写出两种实现方式

```
//方法 1
url = location.href; // var url = 'http://baidu.com?a=1&b=55';
var theRequest = {};
if (url.indexOf("?") != -1) {
    var str = url.substr(url.indexOf("?")+1); //a=1&b=55
    str = str.split("&");
    for(var i = 0; i < str.length; i++) {
        theRequest[str[i].split("=")[0]] = str[i].split("=")[1];
    }
    console.log(theRequest)    //{a: "1", b: "55"}
}

//方法 2
function GetRequest() {
    var url = location.search; // search 取"?"符后的字串 ?a=2&b=33
    var theRequest = {};
    if (url.indexOf("?") != -1) {
        var str = url.substr(1); //a=2&b=33
        str = str.split("&"); //["a=2", "b=33"]
        for(var i = 0; i < str.length; i++) {
            theRequest[str[i].split("=")[0]] = str[i].split("=")[1];
        }
    }
    return theRequest;
}

Request = GetRequest();
console.log(Request) //{a: "2", b: "33"}

//方法 3
// url?a=1&b=2
function getQueryString(name) {
    var reg = new RegExp('(^\&)' + name + '=(^[^&]*)(\&|$)', 'i');
    var r = location.search.substr(1).match(reg);
    if (r != null) {
        return unescape(r[2]);
    }
}
```

```

    }
    return null;
}
console.log(getQueryString('a'))

```

#### 45、ES 5 的继承和 ES 6 的继承有什么区别？

ES5 的继承是通过 prototype 或构造函数机制来实现。ES5 的继承实质上是先创建子类的实例对象，然后再将父类的方法添加到 this 上（Parent.apply(this)）。

ES6 的继承机制完全不同，实质上是先创建父类的实例对象 this（所以必须先调用父类的 super() 方法），然后再用子类的构造函数修改 this。

具体的：ES6 通过 class 关键字定义类，里面有构造方法，类之间通过 extends 关键字实现继承。子类必须在 constructor 方法中调用 super 方法，否则新建实例报错。因为子类没有自己的 this 对象，而是继承了父类的 this 对象，然后对其进行加工。如果不调用 super 方法，子类得不到 this 对象。

#### 46、在金融应用产品中，数值常常使用千分位分隔，请使用 js 实现一个具有此功能的简单常数函数

```

function toNum(value) {
    if (!value) return ' '
    var intPart = Number(value).toFixed(0) // 获取整数部分
    var intPartFormat =
intPart.toString().replace(/(\d)(?=(?:\d{3})+)$/g, '$1,') // 将
整数部分逢三一断
    return intPartFormat;
}
var n = 12001
console.log( toNum(n))//12,001  12001.6->12,002

```

#### 47、webpack 用来解决什么问题

前端本身不支持像后端那样文件引用，使用 webpack 就可以实现这种功能。另外打包还会对代码做检查和压缩，达到优化的目的。

使用 webpack 过程：

- 1、生成 Packjson.json 执行命令 npm init -y;  
文件作用：是 node 的项目描述文件，如项目依赖谁有哪些 scripts 常用
- 2、手动创建 webpack.config.js

#### 48、null 和 undefined 的区别

null 是 javascript 关键字，空值，typeof 返回 object,可以表示数字，字符串和对象“无值”

undefined 是预定义的全局变量，为“未定义”，变量一种取值，表示没有初始化。

当查询对象属性，数组元素值时，返回 undefined 时表示属性或元素不存在；

如果函数没返回值也返回 undefined

#### 49、跟后端进行异步请求时，如何避免代码嵌套太深

现在出现了一种比较好的写法，就是用 Promise.js 来简单来写

```

var asny = function (text) {
    var promise = new Promise(function (resolve, reject) {
        setTimeout(function () {
            console.log(text);
            resolve();
        }, 1000)
    });
};

```

```

        return promise;
    }
    asny("1").then(function () {
        return asny("2");
    }).then(function () {
        return asny("3");
    }).then(function () {
        console.log("done");
    });

```

#### 50、http 协议中 3 0 2 状态是什么含义

该资源原本确实存在，但已经被临时改变了位置,换言之，就是请求的资源暂时驻留在不同的 URI 下

#### 51、new 操作符具体是干什么的

- (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 。

#### 52、如何准确的判断这个变量是否为数组

只有 instanceof 才能判断一个对象是否是真正的数组

#### 53、输出结果 :NaN undefinedhello

```

var a = 10

a.pro = 10
console.log(a.pro + a)//NaN
var s = 'hello';
s.pro = 'world';
console.log(s.pro + s)//undefined hello

```

解释:

变量 a 与 s 都是基本类型，无法给他们添加属性，所以 a.pro 和 s.pro 都是 undefined。

undefined + 10 得到 NaN（not a number）。

undefined + 'hello' 得到 undefinedhello，其中 undefined 被转化为字符串类型。

#### 54、同步和异步的区别

异步操作：同时进行多个操作，用户体验好（如用户名检查，输入完就检查）。异步缺点： 好用但写起来麻烦。

同步操作：一次只能进行一次操作，用户体验不好，按顺序执行,优点：清晰，

#### 55、浏览器输入 URL 到页面在用户面前，这上过程发生了什么

1. 在浏览器中输入网址；
2. 发送至 DNS 服务器并获得域名对应的 WEB 服务器的 IP 地址；
3. 与 WEB 服务器建立 TCP 连接；
4. 浏览器向 WEB 服务器的 IP 地址发送相应的 HTTP 请求；
5. WEB 服务器响应请求并返回指定 URL 的数据，或错误信息，如果设定重定向，则重定向到新的 URL 地址。
6. 浏览器下载数据后解析 HTML 源文件，解析的过程中实现对页面的排版，解析完成后在浏览器中显示基础页面。
7. 分析页面中的超链接并显示在当前页面，重复以上过程直至无超链接需要发送，完

成全部显示。

## 57、返回结果

Typeof 100 返回 number

Typeof (undefined) 返回 undefined

Typeof NaN 返回 number

Typeof (null) 返回 object

## 58、类型检测方法 typeof 和 instanceof 有什么区别

typeof: 返回值是一个字符串, 用来说明变量的数据类型。一般只能返回如下几个结果: number, boolean, string, function, object, undefined。

instanceof: 返回值为布尔值;用于判断一个变量是否属于某个对象的实例。

[instanceof 运算符](#)用于检测构造函数的 [prototype](#) 属性是否出现在某个实例对象的原型链上。

## 59、为什么要对 URL 进行编码

是因为 Url 中有些字符会引起歧义, Url 的编码格式采用的是 ASCII 码

encodeURIComponent 编码的字符范围要比 encodeURI 的大, encodeURIComponent 对特殊字符编码解决用 ASCII 码表示为:3D:= 26:&现在有这样一个问题, 如果我的参数值中就包含=或&这种特殊字符的时候该怎么办。比如说"name1=value1",其中 value1 的值是"va&lu=e1"二手字符串, 那么实际在传输过程中就会变成这样"name1=va&lu=e1"。我们的本意是就只有一个键值对, 但是服务端会解析成两个键值对, 这样就产生了奇异。URL 编码只是简单的在特殊字符的各个字节前加上%, 例如, 我们对上述会产生奇异的字符进行 URL 编码后结果: "name1=va%26lu%3D", 这样服务端会把紧跟在"%"后的字节当成普通的字节, 就是不会把它当成各个参数或键值对的分隔符。

```
console.log(encodeURIComponent("va&lu=e1"))// va%26lu%3De1
```

## 60、vue 生命周期:

beforeCreate 组件实例刚刚被创建,属性都没有

created 实例已经创建完成, 属性已经绑定

beforeMount 模板编译之前 (准备)

mounted 模板编译之后, 代替之前 ready \*

beforeUpdate 组件更新之前 data 数据变了 (用在\$.watch('a',function){})

updated 组件更新完毕 \* (用在\$.watch('a',function){})

beforeDestroy 组件销毁前

destroyed 组件销毁后

Vue 钩子函数: created mounted updated Destroy

## 61、jsonp 原理是什么

1、 其背后原理就是利用了 script 标签不受同源策略的限制, 在页面中动态插入了 script, script 标签的 src 属性就是后端 api 接口的地址,并且以 get 的方式将前端回调处理函数名称告诉后端, 后端在响应请求时会将回调返还, 并且将数据以参数的形式传递回去。

## 62. JavaScript 有哪几种数据类型

参考答案:

简单: Number, Boolean, String, Null, Undefined

引用: Object, Array, Function

## 65、css 选择器优先级？

**! important** ID 选择器 class 选择器 标签选择器 伪类选择器

## 63. document.write 和 innerHTML 的区别

参考答案：

document.write 只能重绘整个页面

innerHTML 可以重绘页面的一部分

## 64、谈谈以前端角度出发做好 SEO 需要考虑什么。

参考答案：

### 1、了解搜索引擎如何抓取网页和如何索引网页

你需要知道一些搜索引擎的基本工作原理，各个搜索引擎之间的区别，搜索机器人（SE robot 或叫 web crawler）如何进行工作，搜索引擎如何对搜索结果进行排序等等。

### 2、Meta 标签优化

主要包括主题（Title），网站描述（Description），和关键词（Keywords）。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。

### 3、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。

### 4、了解主要的搜索引擎

虽然搜索引擎有很多，但是对网站流量起决定作用的就那么几个。比如英文的主要有 Google, Yahoo, Bing 等；中文的有百度，搜狗，有道等。

不同的搜索引擎对页面的抓取和索引、排序的规则都不一样。还要了解各搜索门户和搜索引擎之间的关系，比如 AOL 网页搜索用的是 Google 的搜索技术，MSN 用的是 Bing 的技术。

### 5、主要的互联网目录

Open Directory 自身不是搜索引擎，而是一个大型的网站目录，他和搜索引擎的主要区别是网站内容的收集方式不同。目录是人工编辑的，主要

收录网站主页；搜索引擎是自动收集的，除了主页外还抓取大量的内容页面。

### 6、按点击付费的搜索引擎

搜索引擎也需要生存，随着互联网商务的越来越成熟，收费的搜索引擎也开始大行其道。最典型的有 Overture 和百度，当然也包括 Google 的广告

项目 Google Adwords。越来越多的人通过搜索引擎的点击广告来定位商业网站，这里面也大有优化和排名的学问，你得学会用最少的广告投入获得最

### 7、搜索引擎登录

网站做完了以后，别躺在那里等着客人从天而降。要让别人找到你，最简单的办法就是将网站提交（submit）到搜索引擎。如果你是商业网

站，主要的搜索引擎和目录都会要求你付费来获得收录（比如 Yahoo 要 299 美元），但是好消息是（至少到目前为止）最大的搜索引擎 Google 目前还是免费，而且它主宰着 60% 以上的搜索市场。



## 8、链接交换和链接广泛度 (Link Popularity)

网页内容都是以超文本 (Hypertext) 的方式来互相链接的，网站之间也是如此。除了搜索引擎以外，人们也每天通过不同网站之间的链接来

Surfing ( “冲浪” )。其它网站到你的网站的链接越多，你也就会获得更多的访问量。更重要的是，你的网站的外部链接数越多，会被搜索引擎认为

它的重要性越大，从而给你更高的排名。

## 9、标签的合理使用

### 65、行内元素有哪些？块级元素有哪些？

参考答案：

行内元素主要有：<span>、<a>、<b>、<img>、<br>、<button>、<strong>、<textarea>、<select>

块级元素主要有：<div>、<ul>、<li>、<p>、<fieldset>、<form>、<h1>、<h2>、<h3>、<h4>、<h5>、<h6>、<hr>、

<iframe>、<ol>、<pre>、<table>、<tr>、<td>

行内元素可以通过 display:block 转为块级元素。另外块级元素的 margin 和 padding 都正常，行内元素左右 margin 和左右 padding 正常，上下不识别，

也就是说不能通过 margin-top 和 padding-top 来改变行高

### 66. 如何规避 javascript 多人开发函数重名问题

参考答案：

(1) 可以开发前规定命名规范，根据不同开发人员开发的功能在函数前加前缀

(2) 将每个开发人员的函数封装到类中，调用的时候就调用类的函数，即使函数重名只要类名不重复就 ok

### 67、javascript 面向对象中继承实现

参考答案：

javascript 面向对象中的继承实现一般都使用到了构造函数和 Prototype 原型链，简单的代码如下：

```
function Animal(name) {  
    this.name = name;  
}  
Animal.prototype.getName = function () {  
    alert(this.name)  
}  
function Dog() { };  
Dog.prototype = new Animal("Buddy");  
Dog.prototype.constructor = Dog;  
var dog = new Dog();  
dog.getName(); //Buddy
```

### 68、js 的基础对象有那些，window 和 document 的常用的方法和属性列出来

参考答案：

String, Number, Boolean, Date, Math, Array, RegExp

Window:

方法: setInterval, setTimeout, clearInterval, clearTimeout, alert, confirm, open

属性: name, parent, screenLeft, screenTop, self, top, status

Document:

方法:

createElement, execCommand, getElementById, getElementsByName, getElementByTagName, write, writeln

属性: cookie, doctype, domain, documentElement, readyState, URL

## 69、js 中如何定义 class, 如何扩展 prototype?

参考答案:

Ele.className = "\*\*\*"; //\*\*\*在 css 中定义, 形式如下: .\*\*\* {...}

A.prototype.B = C;

A 是某个构造函数的名字

B 是这个构造函数的属性

C 是想要定义的属性的值

如何添加 html 元素的事件, 有几种方法.

(1) 为 HTML 元素的事件属性赋值

(2) 在 JS 中使用 ele.on\*\*\* = function() {...}

(3) 使用 DOM2 的添加事件的方法 addEventListener 或 attachEvent

## 70、写一段 js, 判断是否是这样组成的: 第一个必须是字母, 后面可以是字母、数字、下划线, 总长度为 5-20

参考答案:

```
var reg = /^[a-zA-Z][a-zA-Z_0-9]{4,19}$/;
```

```
reg.test("ala__ala__ala__ala__");
```

## 71、如何保持浮屠水平垂直居中

(1) 用 position 和负边距: 父节点相对定位。子节 position:absolute;top:50%;margin-身高度的一半; (2)

多行文本居中: vertical-align: middle;display: table-cell; (3) 文本居中 line-height

**Display:flex 弹性布局**, 用来为盒子模型提供最大的灵活性;

CSS 水平垂直居中常见方法总结

### 1、文本水平居中

line-height;text-align:center(文字)

元素水平居中 margin:0 auto

方案 1: position 元素已知宽度

父元素设置为: position: relative;

子元素设置为:

position: absolute; left: 50%;top: 50%;margin: -50px 0 0 -50px;

margin 各减去上下距离的一半

方案 2: position transform 元素未知宽度

子元素: margin: -50px 0 0 -50px;替换为: transform: translate(-50%,-50%);

方案 3: flex 布局

父元素加:

display: flex; //flex 布局

justify-content: center; //使子项目水平居中

align-items: center; //使子项目垂直居中

---

运行结果:

1、结果: x 值是 3, 到 case1 后 x 是 2 在执行 case2

```
var x=0;
switch(++x) {
    case 0:
        ++x;
    case 1:
        ++x;
    case 2:
        ++x;
}
```

---

2、结果: test is not a function

```
var test=1
function test(index){
    console.log(index)
    index=3
}
test(2) //test is not a function
```

解释: var 和 function 同名并不冲突, 但在部分浏览器中, var 会遮蔽同时的 function 导致 function 失效, 只是 test 被赋值就会 not a function, 如 var test 只声明就会显示 2

结果: 1201

此题的关键在于明白 var result = test() 与 var result2 = test() 此处分别产生了两个闭包, 它们之间是独立, 互不干扰的

闭包的产生: 在 A 函数中定义了 B 函数, B 函数中引用了 A 函数中的变量, 此时会产生闭包

```
function addCount() {
    var n = 0
    function add() {
        n++
        console.log(n)
    }
    return { n: n, add: add }
}
var newObj = addCount();
var newObj2 = addCount();
newObj.add()
newObj.add()
console.log(newObj.n)
newObj2.add()
```

---

3、运行 test() 和 new test() 的结果是什么

```
var a = 5;
function test() {
  a = 0;
  console.log(a);
  console.log(this.a);
  var a;
  console.log(a);
}
```

参考答案:

test(): 0 5 0 //这里 this 是 window, 有名函数 this 是全局,  
new test(): 0 undefined 0 //this 是当前对象 test {}

## 附 JS 中的 this 是什么, this 的四种用法

### 1、在一般函数方法中使用 this 指代全局对象 结果: 1

```
function hello() {
  this.x = 1;
  console.log(this.x)////此时控制台打印 1
}
hello();
```

### 2.作为对象方法调用, this 指代上级对象 结果: 1

```
function hello() {
  console.log(this.x) //1
}
var s = {};
s.x = 1;
s.m = hello;
s.m();
// 最后 s={x: 1, m: f}
```

### 3.作为构造函数调用, this 指代 new 出的对象

```
function hello() {
  this.x = 1;
  console.log(this) //hello {x: 1}
}
var s = new hello();
console.log(s.x)
```

### 4、apply 调用, apply 方法作用是改变函数的调用对象, 此方法的第一个参数为改变后

调用这个函数的对象, this 指代第一个参数

```
var x = 0;
function hello() {
```

```

    console.log(this.x)
}

var h = {};
h.x = 1;
h.m = hello;
console.log(h)
h.m.apply(); //输出为 0      this 是 window
h.m.apply(h) //输出 1      this 是 h={}对象
// h={x:1,m:f()}

```

5.

4、参考答案:

```

alert(typeof(null)) -> object
alert(typeof(undefined)) -> undefined
alert(typeof(NaN)) -> number
alert(NaN==undefined) -> false
alert(NaN==NaN) -> false
var str="123abc";alert(typeof(str++)) -> number
alert(str) -> NaN    //number 之后

```

5、结果: 10

```

if('a' in window) {
    var a = 10;
}
console.log(a);    // 10

```

解释: 变量提升、window 的变量

首先, if() {} 的花括号并不像 function() {} 的花括号一样, 具有自己的块级作用域, if 的花括号还是全局的环境。根据 JavaScript 的变量提升机制, var a 会被 js 引擎解释到第一行, 如下:

```

var a;
if ('a' in window) {
    a = 10;
}
} //a 是全局 window.a

```

接着有个知识点, 全局变量是 window 对象的属性, 所以 'a' in window 会返回 true, 答案就很直白了。这道题我在做的时候踩了个坑, 我在代码编辑器里写了如下代码:

```

window.onload = function(){
    if('a' in window){
        var a = 10;
    }
}

```

```
    console.log(a)    // undefined
}
```

这时候，a 这个变量是定义在匿名函数 function() {} 里的，属于该函数的局部变量，所以 a 不再是 window 的对象。大家一定要注意细节。

匿名函数中 if 中变量是局部的，有名函数中 if 中变量是全局的 window

同样也是匿名函数 a 是局部的：undefined

```
(function () {
    if ('a' in window) {
        var a = 10;
    }
    console.log(a); // undefined
})();
```

不是匿名函数 a 是全局的 10

```
function a() {
    if ('a' in window) {
        var a = 10;
    }
    console.log(a); // 10
}
a()
```

6、结果：hello 666666 888888 world!

```
async function sayHello() {
    console.log('Hello')
    await sleep(1000)
    console.log('world!')
}
function sleep(ms) {
    return new Promise(resolve => {
        console.log("666666");
        setTimeout(resolve, ms);
        console.log("888888")
    })
}
sayHello()    // hello 666666 888888 world!
```

解释：

async = true；就表示异步的

async 表示这是一个 async 函数，await 只能用在这个函数里面。

await 表示在这里等待 promise 返回结果了，再继续执行。

首先打出 hello，到了 await，会等待 promise 的返回，所以“world”不会立刻打出，接着进入 sleep 函数，打出 666，接着开了一个 1 秒的定时器，虽然 js 是单线程的，但 setTimeout 是异步的，在浏览器中，异步操作都是被加入到一个称为“events loop”队列的地方，浏览器只会在所有同步代码执行完成之后采

取循环读取的方式执行这里的代码,所以 resolve 被加入任务队列,先打印了 888,一秒后执行了 resolve,表示 promise 成功返回,打出了 world。

### 简要介绍 **Vuex** 原理

**Vuex** 实现了一个单向数据流,在全局拥有一个 **State** 存放数据,当组件要更改 **State** 中的数据时,必须通过 **Mutation** 进行,**Mutation** 同时提供了订阅者模式供外部插件调用获取 **State** 数据的更新。而当所有异步操作(常见于调用后端接口异步获取更新数据)或批量的同步操作需要走 **Action**,但 **Action** 也是无法直接修改 **State** 的,还是需要通过 **Mutation** 来修改 **State** 的数据。最后,根据 **State** 的变化,渲染到视图上。

4、开放性问题爬虫引擎是怎么抓取页面的,如何防止采集?

5、A j a x 的应用场景

8、写出几个H T T P 协议头字段

1 1、不使用 l o o p 循环,创建一个长度为 2 0 的数组,并且每个元素值等于它的下标,计算这个数组中的元素之和

3、js 对 url 进行编码和解码的方法有哪些

encodeURIComponent decodeURI