1. var和let const的区别

1. var是ES5语法，let const是ES6语法，var有变量提升

```
console.log(a) // 终端输出100
var a = 100
```

以上代码经过变量提升，等同于

```
var a
console.log(a) // 终端输出100
a = 100
```

但是对于let const 则不存在变量提升

```
let b = 10
console.log(b) // 终端输出10

console.log(c) // 终端报错 ReferenceError: Cannot access 'c' before initialization
let c = 20
```

2. var和let是变量，可以修改；const是常量，不可修改
3. let const有块级作用域，var没有

```
for(var i = 1; i < 10; i++) {
    var j = i + 1
}
console.log(i,j) // 终端输出10 10

for(let i = 1; i < 10; i++) {
    let j = i + 1
}
console.log(i,j) // 终端报错 ReferenceError: i is not defined.
```

2. typeof 能判断哪些类型

1. 判断值类型 undefined string boolean number symbol
2. 判断引用类型 object (注意typeof null === 'object')
3. 判断函数 function

3. 列举强制类型转换和隐式类型转换

1. 强制类型转换：parseInt, parseFloat, toString等
2. 隐式类型转换：if，逻辑运算，==，+ 拼接字符串

4. 手写深度比较，模拟lodash.isEqual

```
// 实现如下效果
const obj1 = {a:10, b: {x:100,y:100}}
const obj2 = {a:10, b: {x:100,y:100}}
isEqual(obj1,obj2)===true
// 判断是否是对象或数组
```

```
function isObject(obj) {
    return typeof obj === 'object' || obj !== null
}
// 判断是否全相等
function isEqual(obj1, obj2) {
    if(!isObject(obj1) || !isObject(obj2)) {
        // 值类型
        return obj1 === obj2
    }
    if(obj1 === obj2) {
        return true
    }
    // 两个都是对象或数组，而且不相等
    // 1. 先取出obj1和obj2的keys，比较个数
    const obj1Keys = Object.keys(obj1)
    const obj2Keys = Object.keys(obj2)
    if(obj1Keys.length !== obj2Keys.length){
        return false
    }
    // 2. 以obj1为基数，和obj2一次递归比较
    for(let key in obj1){
        // 比较当前key的value是否相等
        const res = isEqual(obj1[key], obj2[key])
        if(!res) {
            return false
        }
    }
    // 3. 全相等
    return true
}
```

5. split()和join()的区别

```
'1-2-3'.split('-') // [1,2,3]

[1,2,3].join('-') // '1-2-3'
```

6. 数组的pop push unshift shift分别做什么

功能是什么，返回值是什么，是否会对原数组造成影响

```
const arr = [10,20,30,40]
const popRes = arr.pop()
console.log(popRes,arr) // 终端输出40, [10,20,30] 返回值是从数组中删除的数组尾部的元素

const pushRes = arr.push(50)
console.log(pushRes, arr) //终端输出4, [10,20,30,50] 返回值是数组的长度

const unshiftRes = arr.unshift(60)
console.log(unshiftRes, arr) // 终端输出5, [60,10,20,30,50] 返回值是数组的长度

const shiftRes = arr.shift()
console.log(shiftRes, arr) // 终端输出60, [10,20,30,50] 返回值是从数组冲删除的数组头部的元素
```

7. 数组的API，有哪些是纯函数?

纯函数的定义：

1. 不改变源数组（改变源数组，表示有副作用）
2. 返回一个数组

```
const arr = [10,20,30,40]

const arr1 = arr.concat([50,60,70])
console.log(arr) // 终端输出[10,20,30,40]
console.log(arr1) // 终端输出[50,60,70]

const arr2 = arr.map((num)=>num*10)
console.log(arr) // 终端输出[10,20,30,40]
console.log(arr2) // 终端输出[100,200,300,400]

const arr3 = arr.filter((num)=>num>25)
console.log(arr) // 终端输出[10,20,30,40]
console.log(arr3) // 终端输出[30,40]

const arr4 = arr.slice() // 类似深拷贝
console.log(arr) // 终端输出[10,20,30,40]
console.log(arr4) // 终端输出[10,20,30,40]
```

纯函数	非纯函数
concat	push pop shift unshift
map	forEach
filter	some every
slice	reduce

8. 数组slice和splice的区别

slice是切片，splice是剪接

slice 是纯函数

```
const arr = [10,20,30,40,50]

const arr1 = arr.slice() // 类似深拷贝
console.log(arr) // 终端输出[10,20,30,40,50]
console.log(arr1) // 终端输出[10,20,30,40,50]

const arr2 = arr.slice(1,4) // 截取index=1到index=4的数组元素
console.log(arr) // 终端输出[10,20,30,40,50]
console.log(arr2) // 终端输出[20,30,40]

const arr3 = arr.slice(2) // 截取index=2到末尾的数组元素
console.log(arr) // 终端输出[10,20,30,40,50]
console.log(arr3) // 终端输出[30,40,50]

const arr4 = arr.slice(-2) // 截取倒数两个数组元素
```

```
console.log(arr) // 终端输出[10,20,30,40,50]
console.log(arr4) // 终端输出[40,50]
```

splice 不是纯函数

```
const arr = [10,20,30,40,50]
const arr1 = arr.splice(1,2,'a','b','c') // 从index=1的位置开始，删除相邻的两个元素，并在此位置添加'a','b','c'三个元素
console.log(arr) // 终端输出 [10,'a','b','c',40,50]
console.log(arr1) // 终端输出 [20,30]
```

9. [10,20,30].map(parseInt)返回结果是什么

map的参数和返回值是什么?

parseInt的参数和返回值是什么?

```
const res = [10,20,30].map(parseInt)
console.log(res) // 终端输出 [10,NaN,NaN]

// 上面的函数相当于
[10,20,30].map((num, index)=>{
    return parseInt(num,index)
}) // 终端输出应该是[parseInt(10,0), parseInt(20,1), parseInt(30,2)]
```

10. ajax里get和post请求的区别

1. get请求一般用于查询，post请求一般用于提交操作
2. get参数拼接在url上，post放在请求体内（数据体积可更大）
3. 安全性: post请求易于防止CSRF攻击

11. 函数call和apply的区别

```
fn.call(this, arg1, arg2, arg3)
fn.apply(this, arguments)
```

12. 事件代理（委托）是什么?

13. 闭包是什么，有什么特性，有什么负面影响?

1. 作用域和自由变量
2. 闭包应用场景：函数作为参数被传入，函数作为返回值被返回
3. 自由变量的查找，要在函数定义的地方（而非执行的地方）

负面影响:

变量会常驻内存，得不到释放，但是并不一定造成内存泄露

14. 如何组织事件冒泡和默认行为

```
event.stopPropagation()
event.preventDefault()
```

15. 查找，添加，删除，移动DOM节点的做法

查找	添加	删除	移动	操作
getElementById	createElement	removeChild	appendChild	setAttribute
getElementsByTagName				getAttribute
querySelector				childNodes
querySelectorAll				parentNode

16. 如何减少DOM操作

为什么要减少DOM操作，因为DOM查询和操作是很消耗性能的

1. 缓存DOM查询结果
2. 多次DOM操作，合并到一次插入

17. 解释JSONP的原理，为何它不是真的ajax

1. 浏览器的同源策略（服务端没有同源策略）和跨域
2. ajax是基于XMLHttpRequest，JSONP是基于script标签可以跨域来实现的
3. 哪些html标签可绕过跨域，img，script，css

jsonp.js里

```
abc({
    name: 'xiayidan'
})
```

18. document load和ready的区别

```
window.addEventListener('load',function(){
    // 页面的全部资源加载完才执行，包括图片和视频等
})
window.addEventListener('DOMContentLoaded',function(){
    // DOM渲染完即可执行，此时图片视频等资源可能还没有加载完
})
```

19. ==和===的区别

== 会尝试类型转换

=== 严格相等

20. 函数声明和函数表达式的区别

函数声明 function() {...}

函数表达式 const fn = function() {...}

函数声明会在代码执行前预加载，而函数表达式不会

21. new Object() 和 Object.create() 的区别

1. {} 等同于 new Object()，原型Object.prototype
2. Object.create(null) 没有原型
3. Object.create({...}) 可指定原型

22. 关于this的场景题

this的值是在执行的时候决定的，而不是在函数定义的时候决定的

```
const User = {
    count: 1,
    getCount: function() {
        return this.count
    }
}
console.log(User.getCount()) // 输出1
const func = User.getCount
console.log(func()) // this是undefined
```

23. 关于作用域和自由变量的场景题 1

```
let i
for(i=1; i < 10; i++){
    setTimeout(function(){
        console.log(i)
    },0)
}
```

24. 判断字符串是以字母开头，后面字母数字下划线，长度6-30

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

// 邮政编码 六位数字
/\d{6}/

// 小写英文字母
/^[a-z]+$/

// 英文字母
/^[a-zA-Z]+$/
```

```
// 日期格式
/\d{4}-\d{1,2}-\d{1,2}$/

// 用户名
/^[a-zA-Z]\w{5,17}$/

// 简单的IP地址匹配
/\d+\.\d+\.\d+\.\d+/
```

25. 关于作用域和自由变量的场景题 2

```
let a = 100
function test() {
    alert(a)
    a = 10
    alert(a)
}
test()
alert(a)
// 100, 10, 10
```

26. 手写字符串trim方法，保证浏览器兼容性

```
String.prototype.trim = function() {
    return this.replace(/^\s+/,'').replace(/\s+$/,'') // 替换命中的片段=>删掉
    // \s是空白字符 从开头开始的空白字符或者从结尾开始的空白字符
}
// 修改String的原型
// this是实例
```

27. 何获取多个数字中的最大值

方法一

```
function max () {
    const nums = Array.prototype.slice.call(arguments) // 将参数变为数组
    let max = 0
    nums.forEach(n=>{
        if(n>max){
            max = n
        }
    })
    return max
}
```

方法二

```
function max () {
    return Math.max(arguments)
}
```

28. 何用JS实现继承

1. class继承
2. prototype继承

29. 如何捕获JS程序中的异常

1. 手动捕获异常 try catch

```
try{
    // todo
} catch (ex) {
    console.error(ex) // 手动捕获catch
} finally {
    // todo
}
```

2. 自动捕获

```
window.onerror = function (message, source, lineNum, colNum, error) {
    // 第一，对跨域的js，如CDN的，不会有详细的报错信息
    // 第二，对于压缩的js，还要配合sourceMap，反查到未压缩代码的行，列
}
```

30. 什么是JSON

1. json是一种数据格式，本质是一段字符串
2. json格式和js对象结构一直，对js语言更友好
3. window.JSON是一个全局对象，JSON.stringify JSON.parse
4. json里面用双引号，属性名也要用双引号括起来

31. 获取当前页面url的参数

1. 传统方式：查找location.search

```
function query (name) {
    const search = location.search.substr(1) // search = 'a=10&b=20&c=30'
    const reg = new RegEx(`(^|&)${name}=([^&]*)(&|$)`,i)
                    // 每个key的名称前面是开头^或者&符号
                    // 每个key的value不能是&符号，但是结尾要跟着&符号或者结尾$
    const res = search.match(reg)
    if(res === null) {
        return null
    }
    return res[2]
}
query('b') // res = ['&b=20&', '&', '20', '&'] 所以返回res[2]
```

2. 新API，URLSearchParams

   缺陷是要做浏览器兼容

```
function query (name) {
    const search = location.search
    const p = new URLSearchParams(search)
    return p.get(name)
}
query('a') // 10
query('b') // 20
```

32. 将url参数解析为js对象

```
// 传统方式，分析search
function queryToObj() {
    const res = {}
    const search = location.search.substr(1) // 去掉前面的?
    search.split('&').forEach(paramStr => {
        const arr = paramStr.split('=')
        const key = arr[0]
        const value = arr[1]
        res[key] = value
    })
    return res
}
```

使用URLSearchParams

```
function queryToObj() {
    const res = {}
    const pList = new URLSearchParams(location.search)
    pList.forEach((val,key)=>{
        res[key] = val
    })
    return res
}
```

33. 手写数组flatern拍平，考虑多层级

```
flat([[1,2],3,[4,5,[6,7,[8,9,[10,11]]]]])
// [1,2,3,4,5,6,7,8,9,10,11]
```

```
function flat(arr){
    // 验证arr中，还有没有深层数组[1,2,[3,4]]
    const isDeep = arr.some(item=>item instanceOf Array)
    if(!isDeep) {
        return arr // 已经是flatern平的[1,2,3,4]
    }
    const res = Array.prototype.concat.apply([],arr)
    return flat(res) // 再递归调用flat方法，继续拍平
}
```

34. 数组去重

传统方式，遍历元素挨个比较，去重

使用Set

考虑计算效率

1. 传统方式

```
function unique(arr) {
    const res = []
    arr.forEach(item=>{
        if(res.indexOf(item)<0) {
            res.push(item)
        }
    })
    return res
}
const res = unique([30,10,20,30,40,10])
console.log(res) // 终端输出10,20,30,40
// 传统方式会比较慢
```

2. 使用Set方法

```
function unique(arr) {
    const set = new Set(arr)
    return [...set]
}
// 使用了set是不允许重复元素存在，并且是无序的，但是数组是有序的
```

35. 手写深拷贝

```
function deepClone(obj={}) {
    if(typeof obj !== 'object' || obj == null ) {
        // obj是null，或者不是对象和数组，直接返回
        return obj
    }
    // 初始化返回结果
    let result
    if(obj instanceOf Array) {
        result = []
    } else {
        result = {}
    }
    for(let key in obj) {
        // 保证key不是原型的属性
        if(obj.hasOwnProperty(key)){
            // 递归调用
            result[key] = deepClone(obj[key])
        }
    }
    // 返回结果
    return result
}
```

注意！Object.assign()不是深拷贝，拷贝的只是obj的第一层级，再往深，就不是深拷贝了

36. 介绍RAF RequestAnimationFrame，性能优化的一部分

1. 要想动画流畅，更新速率要60帧/s，即16.67ms更新一次视图
2. setTimeout要手动控制速率，而RAF浏览器会自动控制
3. 后台标签或隐藏iframe中，RAF会暂停，而setTimeout依然执行，浏览器对RAF做的性能优化

方法一： setTimeout

```
// 3s将宽度从100px变为640px，即增加540px
// 60帧/s，3s就是180帧，也就是每一帧需要增加3px
const $div1 = $(#div1)
let curWidth = 100
const maxWidth = 640

function animate() {
    curWidth = curWidth + 3
    $div1.css('width', curWidth)
    if(curWidth<maxWidth){
        setTimeout(animate,16.7) // 自己控制时间
    }
}
animate()
```

方法二：RAF

```
function animate() {
    curWidth = curWidth + 3
    $div1.css('width', curWidth)
    if(curWidth<maxWidth){
        window.requestAnimationFrame(animate) // 时间不用自己控制
    }
}
animate()
```

37. 前端性能如何优化，一般从哪几个方面考虑?

原则：多使用内存，缓存，减少计算，减少网络请求

方向：加载页面，页面渲染，页面操作流畅度

38. Map和Set 有序和无序

有序：操作慢

无序：操作快

如何结合两者优点? 二叉树及其变种

39. Map和Object的区别

1. API不同
2. Map可以以任意类型为key
3. Map是有序结构（重要），Object是无序结构
4. Map操作同样很快（Map存在的核心价值）

```
const obj = {
    key1: 'value1',
    key2: 100,
    key3: { x: 100 }
}

const m = new Map([
    ['key1', 'value1'].
    ['key2', 100].
    ['key3', { x: 100 }]
])

// Map的基本操作
m.set('name', '夏衣旦')
m.set('key1','hello world')
m.delete('key2')
m.has('key3')
m.forEach((value,key)=>console.log(key.value))
m.size() // 获取map的长度

// Map可以以任意类型为key
const o = {name:'xxx'}
m.set(o,'object key')
// 这个特性的应用场景：希望两个对象有所关联的时候，可以用Map可以接受任意类型的key的特性

// object 有多快?
const obj = {}
for(let i = 0; i < 1000*1000; i++) {
    obj[i+''] = i
}
console.time('obj find')
obj['200000']
console.timeEnd('obj find') // 0.009ms左右
// time和timeEnd方法的入参一样的情况下，会记录两行代码之间的代码的执行时间长度
console.time('obj delete')
delete obj['200000']
console.timeEnd('obj delete') // 0.007ms左右

// Map是有序的，而且操作很快，Map有多快
const m = new Map()
for(let i = 0; i < 1000*1000; i++) {
    m.set(i+'',i)
}
console.time('map find')
obj['200000']
console.timeEnd('map find') // 0.10ms左右
console.time('map delete')
delete obj['200000']
console.timeEnd('map delete') // 0.005ms左右
```

40. Set和Array的区别

1. API不同
2. Set元素不能重复
3. Set是无序结构，操作很快

```
const arr = [10,20,30,40]
const set = new Set([10,20,30,40])
```

```
// Set的基本操作
set.add(50)
set.delete(10)
set.has(20)
set.size()
set.forEach(val=>console.log(val))

// Set的元素不能重复，可以用这个特性来做数组的去重
// Set是无序的（快），因此Set的元素没有Index，无所谓前后，Array是有序的（慢）

// Array都多慢？
const arr = []
for(let i = 0; i < 100 * 10000; i ++) {
    arr.push(i)
}
console.time('arr unshift')
arr.unshift('a')
console.timeEnd('arr unshift') // 1.47ms unshift 会很慢
console.time('arr push')
arr.unshift('b')
console.timeEnd('arr push') // 0.007ms
console.time('arr find')
arr.includes('500000')
console.timeEnd('arr find') // 0.68ms


// Set有多快？
const set = new Set()
for(let i = 0; i < 100 * 10000; i ++) {
    set.add(i)
}
console.time('set add')
arr.unshift('a')
console.timeEnd('set add') // 0.0039ms
console.time('set has')
set.has('500000')
console.timeEnd('set has') // 0.0078ms
```

因此，如果没有有序的需求，可以考虑Set来提升性能。

41. WeakMap和WeakSet

1. 弱引用，防止内存泄漏
2. WeakMap只能用对象作为Key，WeakSet只能用对象做value
3. 没有forEach和size方法，只能用add delete has

```
// WeakMap 弱引用，只能用对象作为key，防止内存泄漏
// 应用场景：为两个对象建立关联关系，并且两者保持独立，并且销毁不受影响
const userInfo = { name: '夏衣旦' }
const cityInfo = { city: '上海' }
const wMap = new WeakMap()
wMap.set(userInfo, cityInfo)
wMap.get(userInfo)

// WeakSet弱引用，只能用对象作为value，防止内存泄漏
const wSet = new WeakSet()
function fn() {
    const obj = { name: '夏衣旦' }
```

```
    wSet.add(obj)
}
fn() // fn()执行完之后，obj对象就会被清除，然后这个时候因为它是弱引用，因此也不会影响它被清除，但是被Set引用的对象，不会被清除，因为这个是强引用
```

42. 数组求和

方法一：传统方式

```
const arr = [10,20,30]
function sum (arr) {
    let res = 0 // 需要新定义一个变量
    arr.forEach(n=>{
      res = res + n
    })
    return res
}
sum(arr) // 输出60
```

方法二：Array.reduce

```
const arr = [10,20,30]
const result = arr.reduce((sum, curNum, index, arr)=>{
    console.log('reduce function ..')
    console.log('sum', sum) // 依次输出 0 10 30 60
    console.log('curNum', curNum) // 依次输出 10 20 30
    console.log('index', index) // 依次输出 0 1 2
    console.log('arr', arr) // 每次输出 [10,20,30]
    return sum + curNum // 返回值会作为下次执行时的第一个参数的值，第一次执行时这个值是0，也就是reduce的第二个参数
},0)
console.log('result', result)

// 写简单一点就是
const res = arr.reduce((sum, curValue) = > sum + curValue, 0)
console.log('res', res)
```

43. Array reduce

1. 数组元素计数，其实就是有条件的累加的过程，不同于上面的数组元素求和，是个无条件的累加的过程

```
const arr = [10,20,30,40,50,10,20,30,20]
// 计算数组中有多少个10多少个20...
const n = 20
const count = arr.reduce((sum, curNum)=> {
    return n === curNum ? sum + 1 : sum
},0)
console.log('count', count)
```

2. 输出字符串

// 传统方式实现

```
const arr = [
    {name: '张三', age: 20},
    {name: '李四', age: 21},
    {name: '小明', age: 22},
]
const str = arr.map(item=>{
    return `${item.name} - &{item.age}`
}).join('\n')
console.log(str)
```

// Array.reduce()方法实现

```
const str = arr.reduce((s,item)=>{
    return `${s}${item.name} - ${item.age}\n`
},'')
console.log(str)
```