前端面试题

javascript面试题

1、javascript有哪些数据类型，它们的区别?

1. **一共8中数据类型**
    1. 基本数据类型(简单数据类型)：string、number、boolean、null、undefined、synbol、bigint
        1. symbol数据类型是ES6中新出的数据类型，创建（Synbol()）之后独一无二并且不可变，不能使用 new Symbol()创建
        2. Bigint数据类型是ES6中新出的数据类型,数据覆盖范围大，能够解决超出普通数据类型范围报错的问题
            1. 创建方法：
                - 整数末尾直接加n：647326483767797n
                - 调用BigInt()构造函数：BigInt("647326483767797")
    2. 复杂数据类型（引用数据类型）：object（对象、数组、正则）
2. **区别:**
    1. 声明变量时的存储分配
        1. 基本数据类型存储在栈中
        2. 复杂数据类型存储在堆中
    2. 不同的访问机制
        1. 基本数据类型可以直接访问
        2. 引用数据类型访问引用地址，根据引用地址找到堆中实体
    3. 复制变量时不同
        1. 基本数据类型：将原始值的副本赋值新的变量
        2. 引用数据类型：将引用地址赋值给新的值

2、闭包?

1. **什么是闭包?**

    方法里返回一个方法

```
function a(){
  let a1 = 1
  return function () {
    return a1
  }
}
```

2. **闭包的意义**
    1. 延长变量的生命周期
    2. 创建私有环境
3. 例如Vue中

```
data(){
  return{

  }
}
```

4. **递归中的闭包面试题**

```
function fn(n,o) {
  console.log(o)
  return {
    fn:function(m){
      return fn(m,n);
    }
  }
}

var a = fn(0) //undefined
console.log(a) //{fn:f}
a.fn(1) // 0
a.fn(2) // 0
a.fn(3) // 0
```

3、函数节流与防抖

1. **函数防抖**

> 函数防抖（debounce），就是指触发事件后，在延迟时间内函数只能执行一次，如果触发事件后在延迟时间内又触发了事件，则会重新计算函数延执行时间。等延迟时间计时完毕，则执行目标代码。（只执行最后一次）

```
封装防抖
//fn 要执行的函数，delay要等待的时间
function debounce(fn, delay) {
  let timeout = null;
  return function () {
    // 清空定时器
    if(timeout !== null) clearTimeout(timeout);
    timeout = setTimeout(()=>{
      fn.call(this)
    }, delay)
  }
}
```

2. **函数节流**

> 函数节流（throttle），在规定时间内，频繁触发的事件被限制为只允许触发一次。（控制高频事件触发次数）

时间戳实现

```
function throttle(fn, delay) {
  let previous = 0;
  // 使用闭包返回一个函数并且用到闭包函数外面的变量previous
  return function() {
      let args = arguments;
      let now = new Date();
      if(now - previous > delay) {
          fn.apply(this, args);
          previous = now;
      }
  }
}
```

定时器实现

```
function throttle(fn,delay) {
  let flag = true
  return function() {
    if(flag) {
      setTimeout(()⇒{
        fn.call(this)
        flay = true
      }, delay)
    }
    flag = false
  }
}
```

4、原型与原型链

1. **原型可以解决什么问题**

   对象共享属性和方法

2. **谁有原型**

   函数：`prototype`

   对象：`__proto__`

3. **对象查找属性或者方法的顺序**

   对象本身⟶构造函数⟶对象的原型⟶构造函数的原型⟶当前原型的原型中

4. **原型链**

   1. 是什么?

      把原型串联起来

   2. 原型链的最顶端是null

5、localStorage、sessionStorage、cookie的区别

1. 共同点

   在客户端存放数据

2. 区别

   1. 存放数据的有效期不同

      sessionStorage：仅在当前浏览器窗口有效，关闭后自动删除

      localStorage：始终有效、窗口或浏览器关闭也会一直存在，除非手动删除

      cookie：只在设置的cookie过期时间内有效

   2. 能过存储的容量不同

      cookie存储量不能超过4k

      localStorage、sessionStorage不能超过5M

注意：不同的浏览器能存储的大小是不同的

6、延迟加载JS有哪些方式?

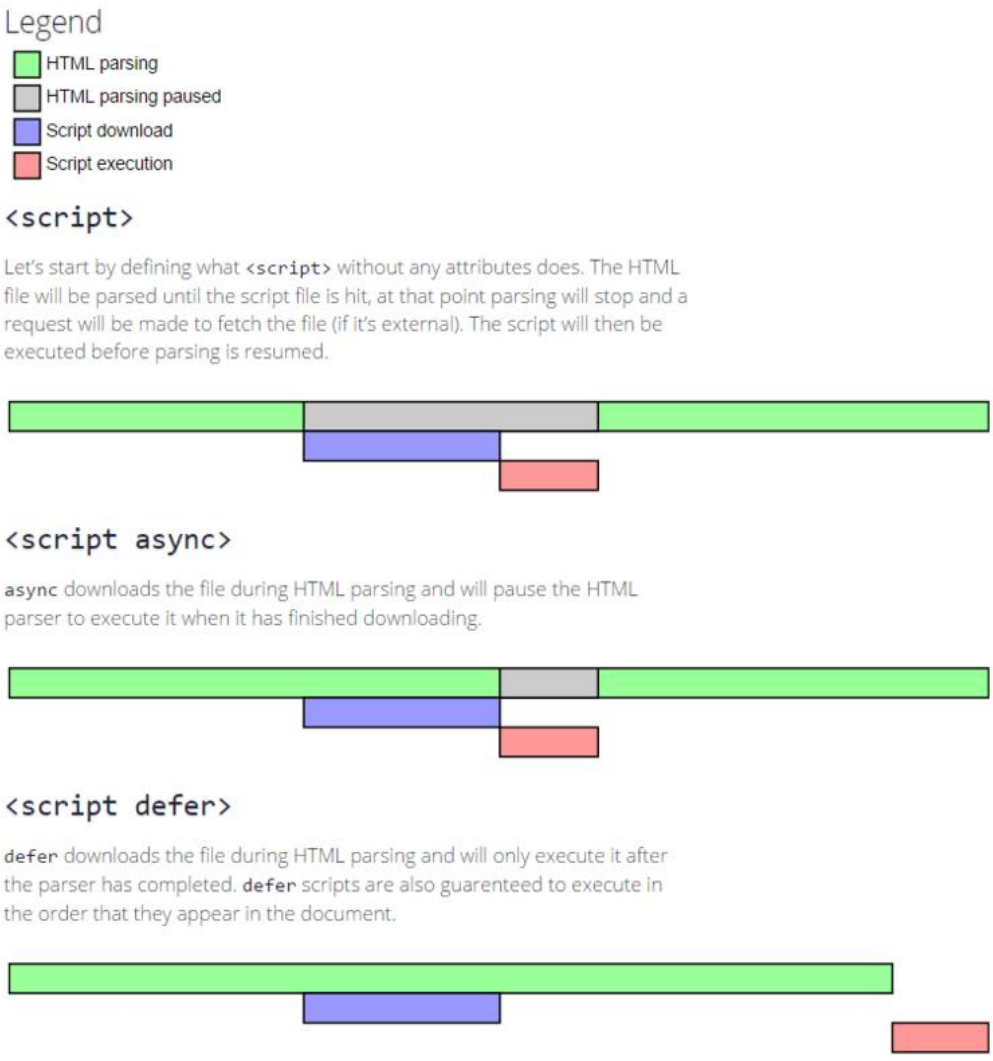
async、

例如：

```
<script async type="text/javascript" src='script.js'></script>
<script defer type="text/javascript" src="script.js"></script>
```

defer:等html全部加载完成，才会执行js代码，顺次执行js脚本

async：async与html解析同步进行，按加载速度执行js脚本



7、null和undefined的区别

null是一个表示“无”的对象（空对象指针），转为数值为0

undefined是一个表示“无”的原始值，转为数值为NaN

8、= 与 == 的区别

= ： 比较值

不同的类型进行比较时会通过调用valueOf转换（JavaScript 调用 `valueOf` 方法将对象转换为原始值。你很少需要自己调用 `valueOf` 方法；当遇到要预期的原始值的对象时，JavaScript 会自动调用它。）

== ： 比较值与类型

9、JS微任务与宏任务

1. js是单线程语言
2. js代码执行流程：

   同步任务⟹事件循环【微任务(清空所有的微任务)⟹宏任务】
3. 事件循环(请求、定时器、事件...)
   1. 微任务：
      1. 新程序或子程序被直接执行
      2. 事件的回调函数
      3. setTimeout()、setInterval()
   2. 宏任务：
      1. Promise.then().catch().finally()
      2. MutationObserver
      3. Object.observe

例如：

```
for(var i = 0; i < 3; i++) {
  setTimeout(function () {
    console.log(i)
  },1000*i)
}
console.log(2222)
//2222
//3
//3
//3

setTimeout(function () {
  console.log('1')
})
new Promise((resolve) => {
   console.log('promise 1')
  resolve()
}).then(() => {
```

```
  console.log('微1')
}).then(() => {
  console.log('微2')
})
console.log(2)

//promise 1
//2
//微1
//微2
//1
```

10、JS作用域

1. 除函数外，js没有块级作用域
2. 作用域链：内部可以访问外部的变量，外部不能访问内部的变量
3. 注意：JS有变量提示机制【变量悬挂声明】
4. 优先级：声明变量 > 声明普通函数 > 参数 > 变量提升

面试时怎么看

1. 本层作用域有没有此变量【变量提升】
2. 注意：js除函数外没有块级作用域
3. 注意：优先级

例题1：

```
function c() {
  var b = 1;
  function a() {
    conso1e.log(b) //undefined
    var b = 2;
    console.log(b) //2
  }
  a();
  console.log(b) //1
  }
c()
```

例题2：

```
var name ='a'
function(){
  if(typeof name ='undefined')
    var name ='b';
    console.log('111'+name);
  }elsef
    console.log('222'+name)
})()
// 111b
```

11、JS作用域+this指向+原型链

例题1:

```
function Foo() {
  getName = function () { console.log(1) }//注意是全局的vindow,
  return this;
}
Foo.getName = function () { console.log(2) }
Foo.prototype.getName = function () { console.log(3) }
var getName = function () { console.log(4) }
function getName() {
  console.log(5)
}
Foo.getName();//2
getName();//4
Foo().getName();//1
//Foo().getName()执行后 相当于在这里执行了 getName = function () { console.log(1) }
getName();//1
new Foo().getName();//3
```

例题2:

```
var o = {
  a:10,
  b:{
    fn:function(){
      console.log(this.a)
      console.log(this)
     }
    }
}
o.b.fn();
//undefined
//{fn:f}
```

例题3：

```
//头条考题
window.name = 'ByteDance'
function A(){
  this.name = 123;
}
A.prototype.getA = function () {
  console.log(this )
  return this.name + 1;
}
let a = new A();
let funcA = a.getA;
/* 相当于 let funcA = function () {
console.log(this)
return this.name + 1;
}*/
funcA() //ByteDance1

//window
```

例题4：

```
var length = 10;
function fn() {
return this.length + 1;
}
var obj = {
  length:5,
  test1:function() {
    return fn();
  }
```

```
}
obj.test2 = fn;
console.log(obj.test1());//11
console.log(fn()===obj.test2());//false
console.log(obj.test1()==obj.test2());//false
```

12.判断变量是不是数组

1. isArray

```
var arr = [1,2,3]
console.log(Array.isArray(arr))
```

2. instanceof

```
var arr = [1,2,3]
console.log(arr instanceof Array)
```

3. 原型 prototype

```
var arr = [1,2,3]
console.log(Object.prototype.toString.call(arr).indexOf('Array'))
```

4. isPrototypeOf()

```
var arr = [1,2,3]
console.log(Array.prototype.isprototypeOf(arr))
```

5. constructor

```
var arr = [1,2,3]
console.log(arr.constructor.toSting().indexOf('Array'))
```

13.slice、splice有什么功能,是否会改变原数组

1. slice用来截取数组

   例如：slice(3)、slice(1,3)、slice(-3)并且会返回一个新数组

2. splice用来：插入、删除、替换

   该方法会改变原数组并返回删除的元素

14.数组去重

1. new Set

```
var arr = [1, 3, 2, 2, 1, 4, 5, 5, 1, 3]
function unique(arr) {
  return [...new Set(arr)]
  // return Array.from(new Set(arr))
}
console.log(unique(arr));
```

2. indexOf

```
var arr = [1, 3, 2, 2, 1, 4, 5, 5, 1, 3]
function unique(arr) {
  var arr1 = []
  for (var i = 0; i < arr.length; i++) {
    if (arr1.indexOf(arr[i]) == -1) {
      arr1.push(arr[i])
    }
  }
  return arr1
}
console.log(unique(arr));
```

3. sort

```
var arr = [1, 3, 2, 2, 1, 4, 5, 5, 1, 3]
function unique(arr) {
  arr = arr.sort()
  var arr1 = []
  for (var i = 0; i < arr.length; i++) {
    if (arr[i] !== arr[i + 1]) {
      arr1.push(arr[i])
    }
  }
  return arr1
}
console.log(unique(arr));
```

15.大数组中包含了4个小数组，分别找到每个小数组中的最大值然后把它们串联起来，形成一个新数组

例:

```
var arr = [
  [4, 5, 1, 3],
  [13, 27, 18, 26],
  [32, 35, 37, 39],
  [1000, 1001, 587, 1]
]
function fnArr(arr) {
  var newArr = []
  arr.forEach((item) => {
    newArr.push(Math.max(...item))
  })
  return newArr
}
console.log(fnArr(arr));
```

16.给字符串对象定义一个addPrefix函数当传入一个字符串str时，它会返回新的带有指定前缀的字符串

例如：console.log("world".addPrefix("hello")) 控制台会输出："helloworld"

```
String.prototype.addPrefix = function (str) {
  return str + this
}
console.log('world'.addPrefix('hello'));
```

17.找出字符串中出现次数最多的字符及其出现的次数

例题1:

```
var str = 'jjjjjjsdfsdijfhaliuyewhjhjhweufghmnnvzxcvj'
function getMaxCount(str) {
  var maxCount = 0
  var maxCountChar = ''
  while (str) {
    var length = str.length
    var char = str.charAt(0)
    var reg = new RegExp(char, 'g')
    str = str.replace(reg, '')
    var newLength = str.length
    var charCount = length - newLength
```

```
    if (charCount > maxCount) {
      maxCount = charCount
      maxCountChar = char
    }
  }
  return {
    maxCount,
    maxCountChar
  }
}
maxCountChar = getMaxCount(str).maxCountChar
maxCount = getMaxCount(str).maxCount
console.log('出现最多的字符为: ' + maxCountChar);
console.log('出现的次数为: ' + maxCount);
```

18.new操作符具体做了什么

1. 创建了一个空的对象
2. 将空对象的原型，指向构造函数的原型
3. 将空对象作为构造函数的上下文（改变this指向）
4. 对构造函数有返回值的处理判断

```
//手写 new
function Fun (age,name) {
  this.age = age
  this.name = name
}
function cerate (fn, ...args) {
  //1. 创建了一个空的对象
  var obj = {}
  //2. 将空对象的原型，指向构造函数的原型
  Object.setPrototypeOf(obj,fn.prototype)
  //3. 将空对象作为构造函数的上下文（改变this指向）
  var result = fn.apply(obj,args)
  //4. 对构造函数有返回值的处理判断
  return result instanceof Object ? result : obj
}
console.log(create(Fun,18,'张三'))
```

19.call、apply、bind的区别

1. 共同点:
    1. 可以改变this指向
    2. 语法: 函数.call()、函数.apply()、函数.bind()

2. 区别

   1. call、apply会立即执行，bind返回的是一个函数而不会立即执行需要加（）才会执行
   2. 参数不同：apply第二个参数为数组的形式,而call和bind有多个参数是需要逐一书写

```
var str= '你好';
var obj = {str:'这是obj对象内的str'}
function fun (name, age){
  this.name = name;
  this.age = age;
  console.log(this,this.str);
}
fun.call(obj); //call立即执行
fun.apply(obj); //apply.立即执行
fun.bind(obj)
bind不会立即执行，因为bind返回的是函数
fun.call(obj,'张三', 88);
fun.apply(obj,['张三', 88]);
fun.bind(obi,'张三', 88))
```

3. 例题

```
<body>
  <button id='btn'>1111</button>
  <h1 id='h1s'>2222</h1>
</body>
<script>
  //1,用apply的情况
  var arr = [1,2,4,5,7,3,321];
  console.log(Math.max.apply(null,arr1))

  //2.用bind的情况
  var btn = document.getElementById('btn');
  var h1s = document.getElementById('h1s');
  btn.onclick = function(){
    console.log(this.id )
  }.bind(h1s)
</script>
```

20.sort

1. 用sort()对[3,15,8,29,102,22]排序

2. 当function(x,y)得到的返回值小于0，x会被移动到y前面(升序排序)

```
 var arr = [1,22,113,12,45,32,334]
 console.log(arr.sort(function(x,y){
   return x - y
 }));
//[1, 12, 22, 32, 45, 113, 334]
```

3. 当function(x,y)得到的返回值大于0，x会被移动到y后面(降序排序)

```
var arr = [1,22,113,12,45,32,334]
console.log(arr.sort(function(x,y){
   return y - x
}));
//[334, 113, 45, 32, 22, 12, 1]
```

4. 当function(x,y)得到的返回值等于0，x和y的位置相对不变（不是所有浏览器都遵守）

21.深拷贝与浅拷贝

1. 共同点：都是复制
2. 区别：
    1. 浅拷贝：只复制引用，不复制值
    2. 深拷贝：复制值

```
var obj = {
  a:1,
  b:2
}
var obj4 = JSON.parse(JSON.stringIfy(obj))
```

22.var、let、const有什么区别

1. var、let声明变量，const声明常量
2. var具有变量提升机制，let和const没有变量提升机制
3. var不支持块级作用域，let和const支持块级作用域
4. var可以重复声明同一变量，let和const不可以重复声明同一变量

23.合并对象

1. Object.assign

```
const a = {a:1, b:4}
const b = {b:2, c:3}
let obj  = Object.assign(a,b)
console.log(obj)
```

2. 拓展运算符（...）

```
const a = {a:1, b:4}
const b = {b:2, c:3}
let obj = {...a, ...b}
const.log(obj)
```

3. 封装方法

```
const a = {a:1, b:4}
const b = {b:2, c:3}
function extend (target, source) {
  for (var key in source) {
    target[key] = source[key]
  }
  return target
}
console.log(extend(a,b))
```

24.箭头函数和普通函数有什么区别?

1. this指向的问题
    - 箭头函数中的this只在箭头函数定义时决定，不可修改（通过call、apply、bind）
    - **箭头函数的this指向定义时外层第一个普通函数的this**
2. 箭头函数不能new（不能当作构造函数）
3. 箭头函数没有prototype
4. 箭头函数没有arguments

25.Promise

1. **Promise有几种状态**
    - 三种状态：pending(进行中)、fulfilled(已成功)、rejected(已失败)
2. **Promise如何解决地狱回调?**
    - 通过链式调用,来防止地狱回调

26.find和filter的区别

1. filter返回一个新的数组
2. find返回匹配到的第一个内容

27.some和every的区别

1. some：数组中任意一项匹配则返回true
2. every：数组中各项全部匹配返回true

```
var arr = [1, 3, 4, 7, 8, 2]
var arr1 = arr.some(val => {
  return val > 4
})
var arr2 = arr.every(val => {
  return val > 4
})
console.log(arr1, arr2) // true false
```

28.数据类型检测的方式有哪些

1. typeof(检测数据类型的运算符)

```
conso1e.log("数值", typeof 10); //number
console.log("布尔", typeof true); //boolean
conso1e.log("字符串", typeof '你好');  //string
console.log("数组", typeof []); //object
console.log("函数", typeof function(){}); //function
console.log("对象", typeof {}); //object
console.log("undefined",typeof undefined); //undefined
console.log("null",typeof null); //object
```

2. instanceof(检测某一个实例是否属于这个类)

```
//可以正确判断对象的类型，不可以判断基本数据类型，内部运行机制，判断在它的原型链上能否找到这个类型的原型。
console.log("数值", 1 instanceof Number); //false
console.log("布尔", true instargeof Boolean); //false
console.log("字符串", '你好'instanceof string); //false
console.log("数组", [] instanceof Array); //true
console.log("函数", function(){} instanceof Function); //true
console.log("对象", {} instanceof object); //true
```

3. constructor(检测实例和类的关系，从而检测数据类型)，引用原来构造该对象的函数

```
console.log("数值", (10).constructor === Number); //true
console.log("布尔", (true).constructor === Boolean); //true
console.log("字符串", ('你好').constructor === String); //true
console.log("数组", ([]).constructor === Array); //true
console.log("函数", (function() {}).constructor === Function); //true
console.log("对象", ({}).constructor === object); //true
```

4. Object.prototype.tostring.call()(检测数据类型)

```
var a = Object.prototype.toString;
console.log("数值", a.call(10); //[object Number]
console.log("布尔", a.call(true)); //[object Boolean]
console.log("字符串", a.call('你好');  //[object string]
console.log("数组", a.call([]); //[object Array]
console.log("函数", a.call(function(){}); //[object Function]
console.log("对象", a.call({}); //[object object]
console.log("undefined", a.call(undefined)); //[object Undefined]
console.log("null", a.call(null)); //[object Null]
```

29.instanceof原理

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。

```
// 定义构造函数
function C(){}
function D(){}

var o = new C();

o instanceof C; // true，因为 Object.getPrototypeOf(o) === C.prototype

o instanceof D; // false，因为 D.prototype 不在 o 的原型链上

o instanceof Object; // true，因为 Object.prototype.isPrototypeOf(o) 返回 true
C.prototype instanceof Object // true，同上

C.prototype = {};
var o2 = new C();

o2 instanceof C; // true

o instanceof C; // false，C.prototype 指向了一个空对象，这个空对象不在 o 的原型链上。

D.prototype = new C(); // 继承
var o3 = new D();
o3 instanceof D; // true
o3 instanceof C; // true 因为 C.prototype 现在在 o3 的原型链上
```

30.typeof NaN的结果是什么?

NaN(not a number)不是一个数字，表示是否属于number类型的一种状态：是或否，不是确切的值

```
console.log(typeof NaN);
var a="abc";
console.1og(Number(a));//NaN,表达式中存在不可转化的变量，返回了无效的结果，不是返回确切的值，
console.log(NaN =NaN);//false NaN不等于本身，不是确切的值，代表一个范围
```

31.“+”操作符什么时候用于字符串的拼接

1. 如果+操作符其中一个乘作数是字符串（或者通过ToPrimitive操作之后最终得到的字符串），则执行字符串的拼接，否则执行数字加法

```
vara={name:"张三"}; //[object ojbect]
//a.valueof()
//a.tostring()
var b={age:18}; //[object ojbect]
//b.valueof()
//b.tostring()
var c = 1;
console.log(a + c);
console.log(a + b); //[object ojbect][object ojbect]
conso1e.log(1 + 1+ '23'); //2+"23"→"223"
cons01e.log(1 + '23' + 4 + 5); //"123"+4+5 → "1234"+5 ⟶ "12345"
```

32.&&（与）和 ||（或）操作符的返回值

1. 首先对第一个操作数进行条件判断，如果不是布尔值，就先强制转换为布尔类型，然后进行条件判断
    1. ||（或）

```
conso1e.log(1 || 2); //1(真)，如果第一个操作数为真，返回第一个操作数的值
conso1e.log(0 || 2); //2(真)，如果第一个操作数为假，返回第二个操作数的值
```

    2. &&（与）

```
conso1e.log(1 && 0);//0(假)，如果第一个操作数为真，返回第二个操作数的值
console.log(0 && 2);//0(假)，如果第一个操作数为假，返回第一个操作数的值
```

33.isNaN和Number.isNaN的区别?

1. isNaN和Number.isNaN:判断一个计算结果或者变量的值是否为NaN
2. isNaN(value):value表示检测的值

    isNaN的判断过程：首先进行类型检测，如果传入的参数不是数值类型，第二步将传入的参数转为数值类型，然后再进行是否为NaN的判断

```
Number('')//0
Number(null)//0
Number(true)//1
Number(false)//0
Number(undefined)//NaN
Number('aa')//NaN
console.log(isNaN(NaN));//true
console.log(isNaN(true));//false
console.log(isNaN('aaa'));//true
console.log(isNaN(null));//false
console.log(isNaN(''));//false
console.log(isNaN(undefined));//true
```

3. `Number.isNaN(value)`:value表示检测的值

   Number.isNaN的判断的过程：首先进行类型检测，如果传入的参数不是数值类型，直接返回false,如果判断是数值类型，然后isNaN()的方式进行判断

```
console.log(Number.isNaN(true)); //false
console.1og(Number.isNaN('')); //false
console.log(Number.isNaN(nu1l)); //false
console.1og(Number.isNaN(NaN)); //true
console.log(Number.isNaN(123)); //false
console.log(Number.isNaN('aaa')); //false
console.log(Number.isNaN(undefined)); //false
```

34.如何判断一个对象是空对象

1. 使用JSON自带的，stringify方法来判断

```
var obj = {name:"张三"}
console.log(JSON.stringify(obj)); //返回一个字符串，{name:"张三"}
if (JSON.stringify(obj) === "{}") {
  console.log("是一个空对象");
}
```

2. 使用object.keys()来判断

```
//key:键名,value:键值
var obj1 = {};
conso1e.log(Object.keys(obj1));//返回一个数组，[]
if(object.keys(obj1).length == 0){
  conso1e.log("是一个空对象")
}
```

35.ES6中的rest参数

用于获取函数多余的参数，形式：(...变量名)，把一个分离的参数序列整合为一个数组

```
function func(...argus){
  console.log(argus); //[1,2,3,4,5,6]
}
func(1,2,3,4,5,6)
```

36.什么是DOM和BOM?

1. **DOM**:document,文档对象类型，用来获取或者设置文档标签的属性
   JS可以通过DOM获取到有哪些标签，标签有哪些属性，内容有哪些
   DOM操作的对象是文档，所以DOM和浏览器没有关系，关注网页本身的内容
2. **BOM**:browser object model,浏览器对象模型，提供独立于内容而与浏览器窗口进行交互的对象
   管理窗口与窗口之间的通讯，核心对象是window⟶location(用于url相关的操作)、history(用于历史相关的操作)，navigator(包含了浏览器相关的信息)

37.arguments类数组

1. 类数组：与数组相似，但是没有数组常见的方法属性
2. 为什么函数的arguments参数是类数组而不是数组？如何遍历类数组?
   1. 利用for循环

      ```
      function func(){
        console.log(arguments);
        for(let i = 0; i < arguments.length; i++) {
         console.log(arguments[i]);
        }
      }
      func(1,2,3,4)
      ```

   2. 将数组的方法应用到类数组上，使用call和apply,使用forEach

      ```
      //call:调用一个对象的一个方法，以另一个对象替换当前的对象
      function func1(){
        Array.prototype.forEach.call(arguments, (item) => {
          console.log(item);
      })
      func1(1,2,3,4,5)
      ```

   3. 将类数组转化成数组，使用Array.from(),对一个类数组，创建一个新的数组实例

```
function func2() {
  const arr = Array.from(arguments)
  arr.forEach((item)⇒{
    console.log(item);
  }
  console.log(arr)
}
func2(1,2,3,4,5)
```

4. 使用展开运算符将类数组转化成数组

```
function func3(){
  const arr1 = [...arguments];
  console.log(arr1);
  arr1.forEach((item) ⇒ {
    console.log(item);
  }
}
func3(11,22,33,44)
```

38.如何判断一个对象是否属于一个类

> 类本身指向就是构造函数，类的数据类型就是函数

```
function Person(name) {
  this.name=name
}
var obj = new Person("张三");
```

1. instanceof:判断构造函数的prototype,属性是否出现在对象的原型链的任何位置

```
var obj1={}
console.log(obj instanceof Person);
console.log(obj1 instanceof Person);
```

2. 对象的属性constructor来判断，指向该对象的构造函数

```
console.log(obj.__proto_.constructor);
```

39.forEach与map的区别

1. 作用：循环遍历数组。
2. 用法：foreach和map的用法，所以这两个的用法基本是一样的。

```
var arr = ['a','b','c'];
arr.forEach(
  (item,index,array)⇒{
  //里面用一个回调函数作为参数，箭头函数里面有三个参数，分别是数组项，数组下标索引，数组本身。
});
console.log(arr);
//通过下标索引修改每一项的值来实现数组的修改。
//foreach和map都用不了break和continue。
```

3. 区别：

   foreach没有返回值，而map有返回值。

40. for...of和for...in的区别

1. for...of遍历获取遍历对象的键值，for...in获取遍历对象的键名
2. 遍历对象：
   1. for...in会遍历对象整个原型链
   2. for..of遍历对象需添加自定义 [Symbol.iterator]() 方法来使该数据结构能够直接被遍历

```
// 通过自定义 [Symbol.iterator]() 方法
function Person(name,age,sex) {
  this.name = name
  this.age = age
  this.sex = sex
}
Person.prototype.height = 188
Var p = new Person("张三", l8,"男")
p[Symbol.iterator] = function() {
  // 初始指针对象指向数组第一个
  let index = 0;
  // 保存 xiaomi 的 this 值
  let _this = this;
  return {
    next: function () {
      // 不断调用 next 方法，直到指向最后一个成员
      if (index < _this.course.length) {
        return { value: _this.course[index++], done: false };
      } else {
        // 每调用next 方法返回一个包含value 和done 属性的对象
        return { value: undefined, done: true };
      }
    }
```

```
  }
}
for(let value of p){
  console.log(value);
}
```

41.[1,2,3] + [4,5,6]的结果为?

```
conso1e.log([1,2,3]+[4,5,6])
//[1,2,3].toString()+[4,5,6].toString()
//1,2,3'+'4,5,6
//结果,"1,2,34,5,6”
```

42.toString()与value0f()的优先级

1. toString()会把数据类型转换成string类型调用数组的toString.方法的时候。其实是调用数组中的每个元素的toString方法，最后将结果串联起来。
2. value0f()方法用于返回指定对象的原始值，若对象没有原始值，则将返回对象本身。
3. 优先级
    1. 在数值运算中，优先使用value0f，
    2. 字符串运算中，优先调用toString
    3. 运算操作符下，value0f优先于tostring，如果value0f无法得到基本类型值，则转用tostring

43.常见跨域解决方式:

1. **JSONP**（html中的script src属性获取其他源的数据）

```
<script>
  function getData(res){
    console.Log(res)
  }
</script>
<script src="http://www.baidu.com/news?callback=getData">
</script>
```

2. **cors**（跨域资源共享支持所有的主流浏览器1e9+）
   XMLHttpRequest发送请求的时候，如果不同源，heaters{Origin}
   后台处理：Access-control-allow-origin:
3. **H5**（window..postMessage跨域）
   主流浏览器ie8+
   window.postMessage("字符串", "*")

44.axios二次封装

1. 配置config/index.js文件

```
const serverConfig = {
  baseURL:'http://localhost:3000/',
  useTokenAuthorization:true,//是否开启 token 认证
}
export default serverConfig
```

2. 配置index.js文件

```
import axios from "axios";
import serverConfig from "./config"
import qs from "qs";

//创建axios实例
let service = axios.create({
  baseURL: serverConfig.baseURL,
  timeout: 3000,
  withCredentials: false, // 跨域请求是否需要携带 cookie
})

// 添加请求拦截器
service.interceptors.request.use(function (config) {
  // 在发送请求之前做些什么
  // 如果开启 token 认证
  if (serverConfig.useTokenAuthorization) {
    config.headers["Authorization"] =
localStorage.getItem("token"); // 请求头携带 token
  }
  // // 设置请求头
  if (!config.headers["content-type"]) { // 如果没有设置请求头
    if (config.method === 'post') {
      config.headers["content-type"] = "application/x-www-
form-urlencoded"; // post 请求
      config.data = qs.stringify(config.data); // 序列化,比如
表单数据
    } else {
      config.headers["content-type"] = "application/json";
// 默认类型
    }
  }
  return config;
}, function (error) {
  // 对请求错误做些什么
  return Promise.reject(error);
```

```
});

// 添加响应拦截器
service.interceptors.response.use(function (response) {
  // 2xx 范围内的状态码都会触发该函数。
  // 对响应数据做点什么
  return response;
}, function (error) {
  // 超出 2xx 范围的状态码都会触发该函数。
  // 对响应错误做点什么
  let message = "";
  if (error && error.response) {
    switch (error.response.status) {
      case 302:
        message = "接口重定向了！";
        break;
      case 400:
        message = "参数不正确！";
        break;
      case 401:
        message = "您未登录，或者登录已经超时，请先登录！";
        break;
      case 403:
        message = "您没有权限操作！";
        break;
      case 404:
        message = `请求地址出错:
${error.response.config.url}`;
        break;
      case 408:
        message = "请求超时！";
        break;
      case 409:
        message = "系统已存在相同数据！";
        break;
      case 500:
        message = "服务器内部错误！";
        break;
      case 501:
        message = "服务未实现！";
        break;
      case 502:
        message = "网关错误！";
        break;
      case 503:
        message = "服务不可用！";
```

```
        break;
      case 504:
        message = "服务暂时无法访问，请稍后再试！";
        break;
      case 505:
        message = "HTTP 版本不受支持！";
        break;
      default:
        message = "异常问题，请联系管理员！";
        break;
    }
  }
  return Promise.reject(message);
});

export default service
```

45.CSRF攻击是什么?

1. 概念：跨域请求伪造
2. 原理：诱导用户跳转到新的页面，利用服务器的验证漏洞和用户之前的登陆状态，来模拟用户进行操作。
3. 防范：
    1. 利用cookie的sameSize属性规定其他网站不能使用本站cookie
    2. 使用token验证。再去验证用户身份

46.XSS攻击是什么?

1. 概念：跨站脚本攻击
2. 原理：攻击者通过向被攻击网站注入恶意代码实现攻击，当被攻击者登录网站就会执行恶意代码，读取cookie、、session以及其他敏感信息，对用户进行钓鱼欺骗

47.说一说创建ajax过程?

1. 创建XMLHttpRequest对象,创建一个异步调用对象.
2. 创建一个新的HTTP请求,并指定该HTTP请求的方法、 URL及验证信息.
3. 设置响应HTTP请求状态变化的函数.
4. 发送HTTP请求

HTML+CSS面试题

1.行内元素有哪些？块级元素有哪些？空（void）元素有哪些？

1. 行内元素（不独占一行，且不能设置宽高）：span、img、input...
2. 块级元素（独占一行，且可以设置宽高）：div、p、h1—h6、footer、header、section...
3. 空元素：br、hr..
4. 元素之间的转换问题：
    - `display: inline;` 把某元素转换成行内元素
    - `display: block;` 把某元素转换成块元素
    - `display: inline-block;` 把某元素转换为行内块元素（不独占一行，并且可以设置宽高）

2.页面导入样式时，使用link和@import有什么区别？

1. link的兼容性比@import好
2. 浏览器先加载link标签，后加载@import

3.title与h1、b与strong、i与em之间的区别？

1. **title与h1的区别：**
    1. 定义：
        - title：网站标题
        - h1：网站内容标题
    2. 区别：
        - title显示在网页的标题上，h1显示在网页内容上
2. **b与strong的区别：**
    1. 定义：
        - b：实体标签，用来给文字加粗
        - strong：逻辑标签，用来加强语气
    2. 区别：
        - b标签只加粗样式，没有实际含义
        - strong标签表示标签内字符比较重要，用以强调
3. **i与em的区别：**
    1. 定义：
        - i：实体标签，用来做文字倾斜
        - em：逻辑标签，用来强调文字内容
    2. 区别：

- i：没有实际含义
- em：表示标签内字符比较重要，用以强调

4.img标签中title和alt有什么区别?

1. title：鼠标移入到图片显示的值
2. alt：图片无法加载时显示的值
3. 在seo层面，爬虫抓取不到图片的内容，所以前端在写img标签的时候为了增加seo效果要加入alt属性来描述这张图片的内容或关键词

5.png、jpg、webp、gif这些图片格式有什么区别，分别什么时候用?

1. png：无损压缩，尺寸体积比jpg/jpeg大，适合做小图标
2. jpg：采用压缩算法，会产生失真，比png体积小，适合做中大图片
3. webp：同时支持有损或无损压缩，相同质量的图片webp具有更小的体积，但兼容性低

6.语义化标签

1. 常用的新增语义化标签：header(头部标签)、nav（导航标签）、article(内容标签)、setion(定义文档某个区域)、aside(侧边栏标签)、footer(尾部标签)
2. 特点：
    1. 易读性和维护性更好
    2. seo成分更好，更好爬取
    3. IE8不兼容

7.::before和:after中双冒号和单冒号有什么区别？这两个伪元素有什么用

1. 区别：

    :是伪类、 ::是伪元素
2. 作用：

    清楚浮动、样式布局

8.如何关闭IOS键盘首字母自动大写

input标签添加autocapitalize='off'属性

9.怎么让Chrome支持小于12px的文字

1. Chrome默认字体大小为16px，支持最小字体大小为12px
2. 添加`-webkit-transform:scale(1.6);`缩放属性

10.rem与em的区别

1. 使用rem单位的时候，页面转换为像素大小取决于**根元素的字体大小，即HTML元素的字体大小**。根元素字体大小乘rem的值。
2. 当使用em单位的时候，像素值是将**em值乘以使用em单位的元素的字体大小**。当使用em的单位没有设置字体大小时，将会根据**父元素的字体大小计算**。

11.rem适配

1rem的大小就是根元素的font-size的值。通过设置根元素的font-size的大小来控制整个HTML文档内的文字大小、元素宽高、内外边距等。根据移动设备的宽度大小来实现适应，不同的设备都展示一致的页面效果。

12.响应式

1. 什么是响应式?

   同一个URL可以响应多端

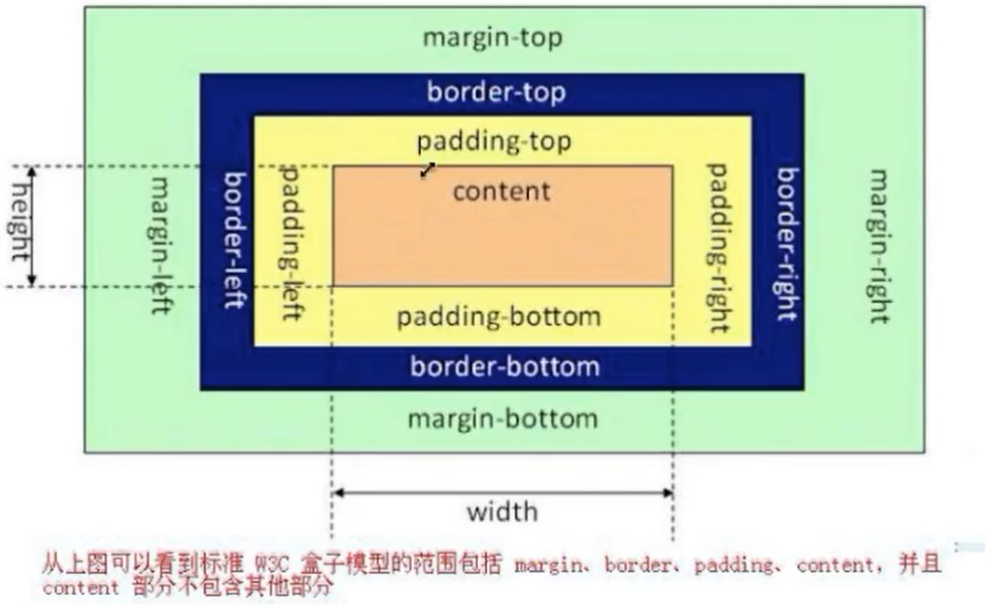
2. 语法:

```
@media only screen and (max-width:1000px) {

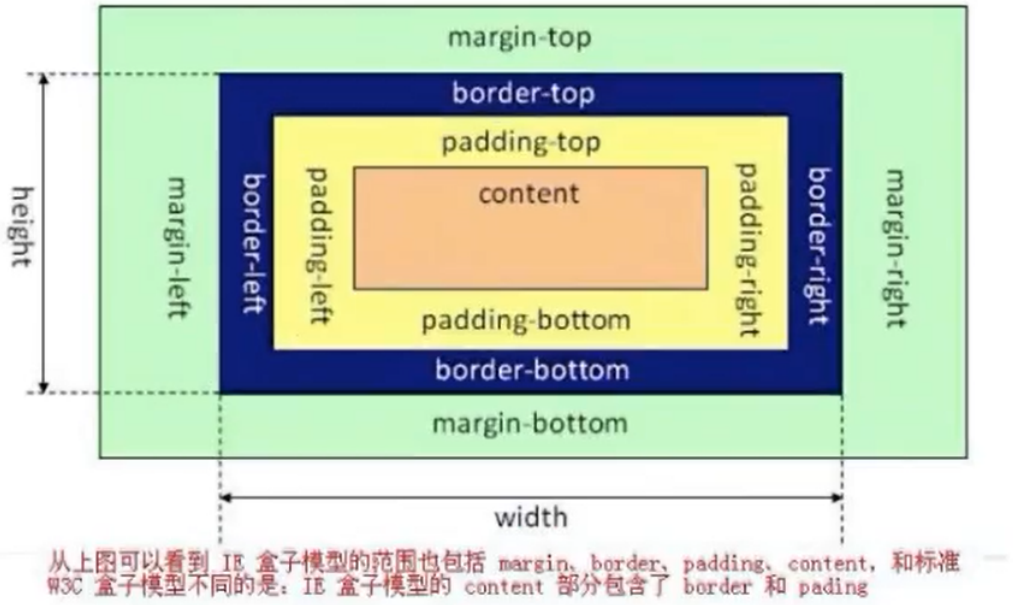
}
only:可以排除不支持媒体查询的浏览器
screen: 设备类型
max-width | max-height
min-width | min-height
```

13.介绍一下CSS盒子模型

1. CSS盒子模型包括:
   1. 标准模型



2. IE盒子模型



2. 组成：
   - 标准盒子模型：margin、border、padding、content
   - IE盒子模型：margin、content (border + padding + content)
3. 通过CSS转换盒子模型：
   - `box-sizing: content-box;`标准盒子模型
   - `box-sizing: border-box;`IE盒子模型

14. line-height和height的区别

1. **height**是盒子的高度
2. **line-hright**是每一行文字的高度，会改变盒子的高度

15.CSS选择符有哪些？那些属性可以继承？

1. CSS选择符：
    - 通配符（*）、id选择器（#）、类选择器（.）、标签选择器（div、p、span...）、相邻选择器（+）、后代选择器（ul li）子元素选择器（>）、属性选择器（div[]）
    - ```
      相邻选择器（+）选择下一个兄弟
      div + li{}
      ```
2. 那些属性可以继承
    - font-size、color、line-height ...
3. 不可以继承
    - border、padding、margin ...

16.CSS优先级

! important > 内联样式 > id > class > 标签/伪元素选择器 > 通配符

17.纯CSS画三角形

```
div{
  width: 0;
  height: 0;
  border: 125px solid transparent;
  border-bottom-color: pink;
}
```

18.不设置宽度高度如何给盒子设置垂直居中

1. 利用display: flex
2. 利用定位（**子绝父相**）+transform: translate(-50%, -50%)

19.dispaly有哪些值？说明它们的作用

1. none （隐藏元素）
2. block （将元素转换为块元素）
3. inline （将元素转换为行内元素）
4. inline-block （将元素转换为行内块元素）
```

20.对BFC的理解

1. BFC (block formatting context) 块级格式化上下文
2. BFC原则：如果一个元素具有BFC ，那么其内部元素将不影响外面的元素
3. 如何触发BFC：
    - float的值为非none
    - overflow的值为非visible
    - display的值为：inline-block、table-cell...
    - position的值为：absoute、fixed...

21.清除浮动

1. 触发BFC
2. 添加伪元素

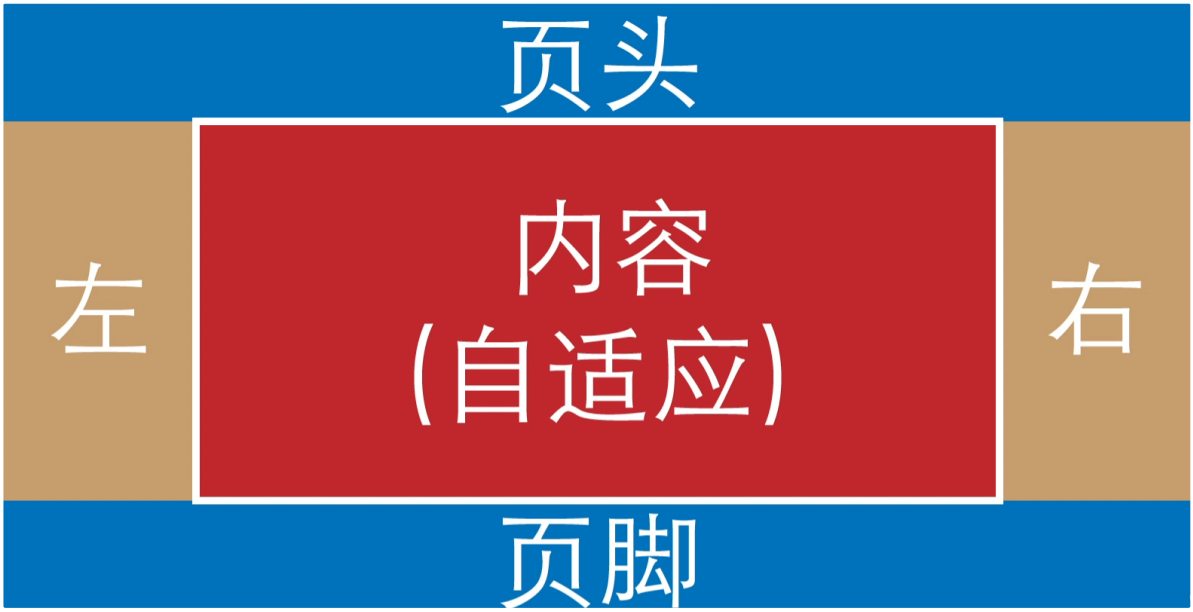
```
:after{
  content: '';
  display: block;
  clear: both;
}
```

3. 多创建一个盒子，添加样式：clear: both;

22.定位

值	描述
static	默认值，无定位
absolute	绝对定位，根据第一个有relative属性的父元素进行定位，脱离文档流
fixed	固定定位，相对于浏览器窗口进行定位
relative	相对定位，相对于原位置进行定位，不脱离文档流。当**left、right、top、bottom**同时存在时以**left、top**为准

23.圣杯布局与双飞翼布局



1．圣杯布局

```
<boby>
  <header>头部</header>
  <div class="clearfix wrapper">
    <div class="center">主区域</div>
    <div class="left">左区域</div>
    <div class="right">右区域</div>
  </div>
</boby>

<style>
  .left, .right, .center {
    float: left;
  }
  .clearfix:after {
    content: '';
    display: block;
    clear: both;
  }
  .wrapper {
    padding: 0 100px;
  }
  .left {
    position: relative;
    left: -100px
    margin-left: -100%;
    width: 100px;
  }
  .right {
    position: relative;
    right: -100px;
```

```
    margin-left:-100px;
    width: 100px
  }
  .center {
    width: 100%;
  }
</style>
```

2. 双飞翼布局

```
<boby>
  <div class="header">头部</div>
  <div class="wrapper">
    <div class="center">主区域</div>
  </div>
  <div class="left">左区域</div>
  <div class="right">右区域</div>
</boby>

<style>
  .wrapper {
    float: left;
    width: 100%;
  }
  .left {
    float: left;
    margin-left: -100%;
    width: 100px;
  }
  .right {
    float: left;
    margin-left: -100px;
    width: 100px
  }
  .center {
    margin: 0 100px;
  }
</style>
```

24.什么是CSS reset

1. reset.css 是一个用来重置css样式的文件
2. normalize.css 为了增强跨浏览器渲染的一致性的CSS重置样式库

25.什么是css sprite（精灵图），有什么优缺点?

1. 精灵图是什么

   把多个小图标合成一张大图片

2. 优缺点

   - 优点：减少了http请求的次数，提升了性能
   - 缺点：可维护性差

26.display:none;与visiblity:hidden;的区别

1. 占用位置的区别

   - display: none; 不占位置
   - visiblity: hidden; 占用位置

2. 重绘和回流

   display: none;:产生一次回流、一次重绘

   visiblity: hidden;:产生一次重绘

   产生回流的情况：改变元素的位置（left、top...）、显示隐藏元素...
   产生重绘的情况：样式改变...

27.opacity和rgba的区别

1. 共同：实现透明效果

   - opacity：取值在0~1之间，0表示透明，1表示不透明
   - rgba：R表示红色，G表示绿色，B表示蓝色，A表示透明度取值在0~1之间

2. 区别：

   - opacity的子代会继承父元素的透明属性
   - rgba的子代不会继承父元素的透明属性

28.图片懒加载

1. getBoundingClientRect API + Scroll

   Element.getBoundingClientRect() 方法返回元素的大小及其相对于视口的位置。

```
<body>
  <img data-src="1.jpg">
  <img data-src="1.jpg">
  <img data-src="1.jpg">
</body>
<script>
  const images document.querySelectorAll('img');
  window.addEventListener('scroll',(e)⇒{
    images.forEach(image ⇒{
```

```
      const imageTop = image.getBoundingclientRect().top;
      if(imageTop < window.innerHeight){
        const data_src = image.getAttribute('data-src');
        image.setAttribute('src',data_src);
      }
      console.log('scrollf触发"):
    })
  });
</script>
```

2. IntersectionObserver API + DataSet API
   1. **IntersectionObserver API，能够监听元素是否到了当前视口的事件**

```
<body>
  <img data-src="1.jpg">
  <img data-src="1.jpg">
  <img data-src="1.jpg">
</body>
<script>
  const images document.querySelectorAll('img');
  const callback = entries => {
    entries.forEach(entry =>{
      if(entry.isIntersecting) {
        const image = entry.target;
        const data_src = image.getAttribute('data-src')
        observer.unobserve(image)
        image.setAttribute('src',data_src);
      }
    })
  }
  const observer.=new IntersectionObserver(callback )
  images.forEach(image) => {
    observer.observe(image)
  }):
</script>
```

Vue面试题

1.生命周期

1.1Vue2

1. **11个生命周期钩子**
    1. 将要创建(`beforeCreate`)、创建完毕(`created`)、将要挂载(`beforeMount`)、**挂载完毕(`mounted`)**、将要更新(`beforeUpdate`)、更新完毕(`updated`)、**将要销毁(`peforeDestroy`)**、销毁完毕(`destroyed`)、下一次DOM更新结束后执行(`$nextTick(回调函数)`)
    2. **路由组件所独有的两个生命周期钩子**
        1. 路由组件被激活时触发(`activated`)、路由组件失活时触发(`deactivated`)
        2. 作用：用于捕获路由组件的激活状态。
2. **常用的生命周期钩子**：
    1. mounted：发送ajax请求、启动定时器、绑定自定义事件、订阅消息等【初始化操作】。
    2. beforeDestroy：清除定时器、解绑自定义事件、取消订阅消息等【收尾工作】。

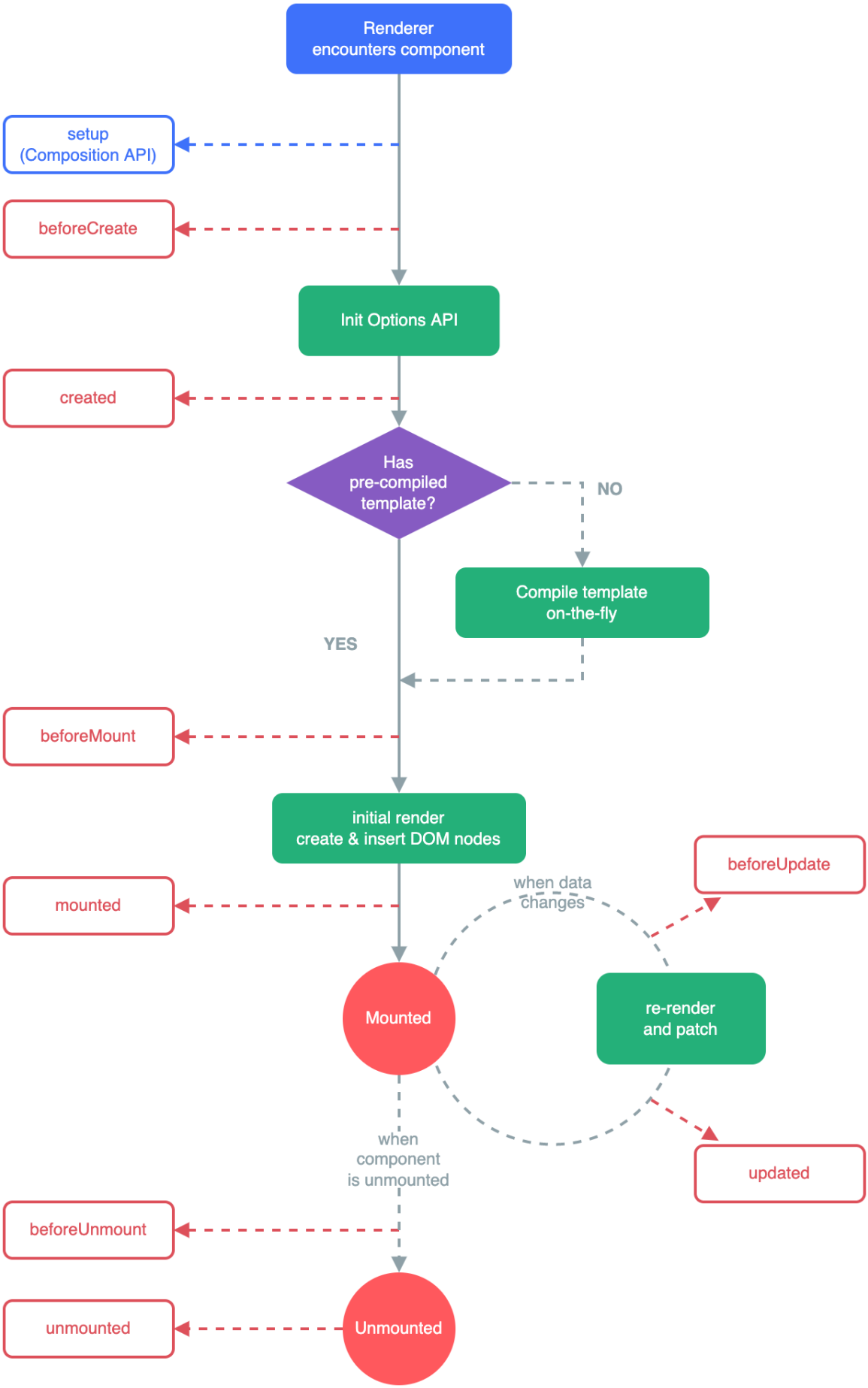
1.2.Vue3

1. Vue3.0中可以继续使用Vue2.x中的生命周期钩子，但有有两个被更名：

- `beforeDestroy`改名为 `beforeUnmount`
- `destroyed`改名为 `unmounted`

2. Vue3.0也提供了 Composition API 形式的生命周期钩子，与Vue2.x中钩子对应关系如下：

- `beforeCreate`===>`setup()`
- `created`===>`setup()`
- `beforeMount` ===>`onBeforeMount`
- `mounted`===>`onMounted`
- `beforeUpdate`===>`onBeforeUpdate`
- `updated` ===>`onUpdated`
- `beforeUnmount` ==>`onBeforeUnmount`
- `unmounted` ===>`onUnmounted`


Renderer encounters component
setup (Composition API)
beforeCreate
Init Options API
created
Has pre-compiled template?
NO
Compile template on-the-fly
YES
beforeMount
initial render create & insert DOM nodes
mounted
beforeUpdate
when data changes
Mounted
re-render and patch
updated
when component is unmounted
beforeUnmount
unmounted
Unmounted

2.keep-alive 缓存路由组件

1. 作用：让不展示的路由组件保持挂载，不被销毁。
2. 具体编码：

```
<keep-alive include="News">
    <router-view></router-view>
</keep-alive>
```

3.$nextTick是什么

1. 语法：this.$nextTick(回调函数)
2. 原理：返回一个Promise
3. 作用：在下一次 DOM 更新结束后执行其指定的回调。
4. 什么时候用：当改变数据后，要基于更新后的新DOM进行某些操作时，要在nextTick所指定的回调函数中执行。

4.v-if和v-show之间的区别

1. 方式：
    1. v-if 操作DOM
    2. v-show 操作CSS样式
2. 编译方式：
    1. v-if：DOM编译和卸载的过程
    2. v-show：基于CSS
3. 性能与使用场景：
    1. v-if 性能消耗高，适合执行相对复杂的任务
    2. v-show 性能消耗低，适合频繁切换的任务

5.ref属性

1. 被用来给元素或子组件注册引用信息
2. 应用在html标签上获取的是真实DOM元素，应用在组件标签上是组件实例对象
3. 使用方式：
    1. 打标识：<h1 ref="xxx">.....</h1> 或 <School ref="xxx"></School>
    2. 获取：this.$refs.xxx

6.scoped样式

1. **作用**：让样式在局部生效，防止冲突。
2. **原理**：给节点新增自定义属性，然后CSS根据属性选择器添加样式
3. **写法**：`<style scoped>`

7.样式穿透

1. **css、stylus**样式穿透：`::v-deep、/deep/、 >>>`
2. **less、sass**样式穿透：`::v-deep、/deep/`

8.组件间通信

1. 父组件传值到子组件(**props**)
    1. 功能：让组件接收外部传过来的数据
    2. 传递数据：`<Demo name="xxx"/>`
    3. 接收数据：
        1. 第一种方式（只接收）：`props:['name']`
        2. 第二种方式（限制类型）：`props:{name:String}`
        3. 第三种方式（限制类型、限制必要性、指定默认值）：

```
props:{
  name:{
      type:String, //类型
      required:true, //必要性
      default:'老王' //默认值
  }
}
```

> 备注：props是只读的，Vue底层会监测你对props的修改，如果进行了修改，就会发出警告，若业务需求确实需要修改，那么请复制props的内容到data中一份，然后去修改data中的数据。

2. 子组件传值到副组件
    1. 子组件：

        `this.$emit("自定义事件名",要传的数据)`

    2. 父组件：

```
<Header @自定义事件名='方法'></Header>
methods: {
  方法(传过来的数据) {
    //处理传过来的数据
  }
}
```

3. 任意组件间通信
    1. 全局事件总线（GlobalEventBus）
        1. 一种组件间通信的方式，适用于任意组件间通信。
        2. 安装全局事件总线：

```
new Vue({
  ......
  beforeCreate() {
    Vue.prototype.$bus = this //安装全局事件总线，$bus就是当前应用的vm
  },
    ......
})
```

        3. 使用事件总线：
            1. 接收数据：A组件想接收数据，则在A组件中给$bus绑定自定义事件，事件的回调留在A组件自身。

```
methods(){
  demo(data){......}
}
......
mounted() {
  this.$bus.$on('xxxx',this.demo)
}
```

            2. 提供数据：this.$bus.$emit('xxxx',数据)
        4. 最好在beforeDestroy钩子中，用$off去解绑当前组件所用到的事件。

9. computed与methods的区别

1. computed有缓存
2. methods没有缓存

10.computed与watch区别

1. computed能完成的功能，watch都可以完成。
2. watch能完成的功能，computed不一定能完成，例如：watch可以进行异步操作。
3. 两个重要的小原则：
    1. 所有被Vue管理的函数，最好写成普通函数，这样this的指向才是vm 或 组件实例对象。
    2. 所有不被Vue所管理的函数（定时器的回调函数、ajax的回调函数等、Promise的回调函数），最好写成箭头函数，

        这样this的指向才是vm 或 组件实例对象。

11.props和data优先级谁高?

props ⟹ methods ⟹ data ⟹ computed ⟹ watch

12.Vuex面试题

12.1.Vuex有哪些属性?

1. state、getters、mutations、actions、modules
2. 作用：
    1. state类似于组件中data,存放数据
    2. getters类型于组件中computed
    3. mutations类似于组件中methods
    4. actions提交mutations的
    5. modules把以上4个属性再细分，让仓库更好管理

12.2.Vuex是单向数据流还是双向数据流?

Vuex是单向数据流

12.3.vuex中的mu taitons和actions区别

1. mutaitons:都是同步事物
2. actions:可以包含任意异步操作

12.4.Vuex如何做持久化存储

Vuex本身不是持久化存储
1.使用localStorage
2.使用Vuex-persist插件

13.Vue路由

13.1.两种路由模式

1. hash值不会包含在 HTTP 请求中，即：hash值不会带给服务器。
2. hash模式：
    1. 地址中永远带着#号，不美观 。
    2. 不会发送请求
    3. 兼容性较好。
3. history模式：
    1. 地址干净，美观 。
    2. 兼容性和hash模式相比略差。
    3. 会发送请求，应用部署上线时需要后端人员支持，解决刷新页面服务端404的问题。

13.2.路由传值

1. 显式(query参数)
    1. 传递参数

```
<!-- 跳转并携带query参数，to的字符串写法 -->
<router-link :to="/home/message/detail?id=666&title=你好">跳转</router-link>

<!-- 跳转并携带query参数，to的对象写法 -->
<router-link
  :to="{
    path:'/home/message/detail',
    query:{
       id:666,
       title:'你好'
    }
  }"
>跳转</router-link>
```

    2. 接收参数：

```
$route.query.id
$route.query.title
```

2. 隐式（params参数）
    1. 配置路由，声明接收params参数

```
{
  path:'/home',
```

```
  component:Home,
  children:[
    {
      path:'news',
      component:News
    },
    {
      component:Message,
      children:[
        {
          name:'xiangqing',
          path:'detail/:id/:title', //使用占位符声明接收params参数
          component:Detail
        }
      ]
    }
  ]
}
```

2．传递参数

```
<!-- 跳转并携带params参数，to的字符串写法 -->
<router-link :to="/home/message/detail/666/你好">跳转</router-link>

<!-- 跳转并携带params参数，to的对象写法 -->
<router-link
  :to="{
    name:'xiangqing',
    params:{
       id:666,
       title:'你好'
    }
  }"
>跳转</router-link>
```

> 特别注意：路由携带params参数时，若使用to的对象写法，则不能使用path配置项，必须使用name配置!

3．接收参数：

```
$route.params.id
$route.params.title
```

13.3.路由守卫

1. 作用：对路由进行权限控制
2. 全局守卫
    1. （router.beforeEach）全局前置守卫：初始化时执行、每次路由切换前执行
    2. （router.afterEach）全局后置守卫：初始化时执行、每次路由切换后执行
3. 独享守卫
    1. beforeEnter
4. 组件内守卫
    1. （beforeRouteEnter）进入守卫：通过路由规则，进入该组件时被调用
    2. （beforeRouteLeave）离开守卫：通过路由规则，离开该组件时被调用

13.4.动态路由

1. 场景：详情页
2. router.js配置：

```
{
  path:"/list",
  name:"List",
  children:[
        {
          path:"list/:id",
          name:"Details",
          component: () => import("../views/Details.vue")
        }
  ],
  component: () => import("../views/list.vie")
}
```

14.SPA是什么？有什么缺点

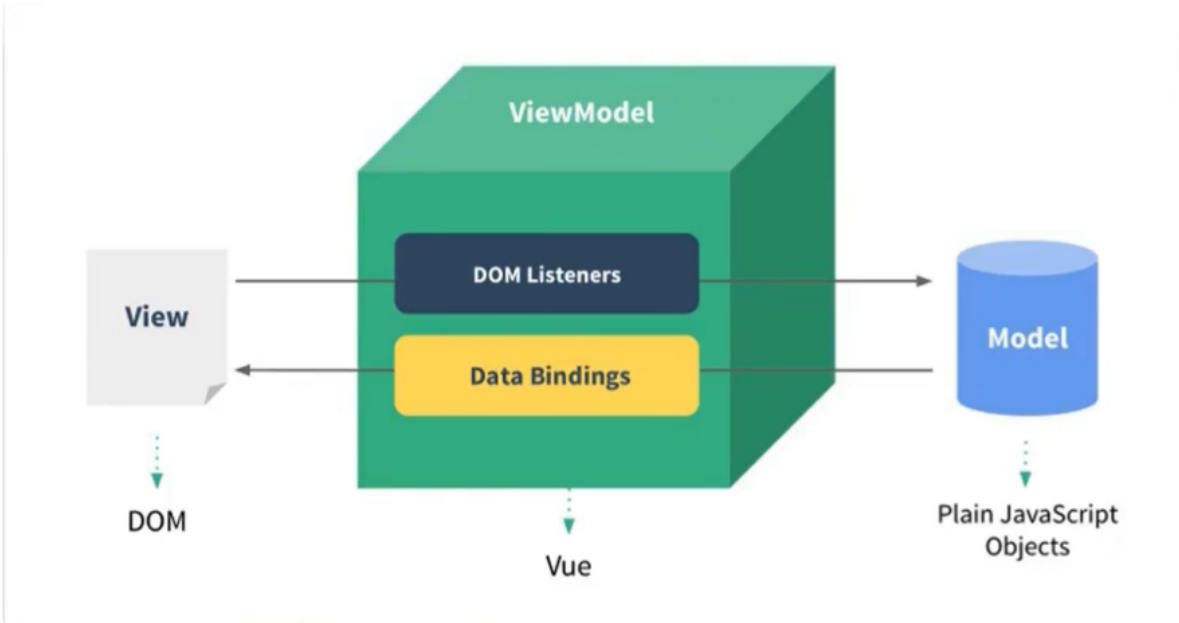
1. SPA是什么？
    - 单页面应用
2. 缺点：
    1. SEO优化不好
    2. 性能一般

15.双向绑定原理

通过Object.defineProperty(数据代理)劫持数据发生的改变，如果数据发生改变了（在set中进行赋值的），触发update方法
进行更新节点内容({{str}})，从而实现了数据双向绑定的原理。

16.MVVM模型

1. M：模型(Model) ：data中的数据
2. V：视图(View) ：模板代码
3. VM：视图模型(ViewModel)：Vue实例



17.v-for和v-if为什么不建议一起使用?

v-for优先级高于v-if，将两者放在一起，会先执行v-for再执行v-if，造成性能浪费

18.配置vue.config.js解决跨域问题

1. 浏览器的同源策略：就是两个页面具有相同的**协议(protocol)、主机(host)、端口号(port)**
2. 请求一个接口时，出现Access-Control-Allow-0 rigin等就说明请求跨域了
3. vue中解决跨域：配置vue.config.js文件
    1. 原理：
        1. 将域名发送给本地的服务器(1oclahost:8080)
        2. 再由本地的服务器去请求真正的服务器
        3. 因为请求是从服务端发出的，所以就不存在跨域的问题了
        4. 注意：修改vue.config.js文件需要重启服务
    2. vue.config.js

```
module.exports = {
  devServer:{
    /跨域
    proxy:{
      '/api':{
        //目标路径
        target:'https://www.bilibili.com/',
        //允许跨域
        changeOrigin:true,
        //重写路径
        pathRewrite:{
          '^/api':''
        }
      }
    }
  }
}
```

19.Vue首页白屏的解决方案

1. **骨架屏**
   骨架屏就是在进入项目的FP阶段，给它来一个类似轮廓的东西
2. **loading**
   首页加一个loading:在index.html里加一个loading css效果，当页面加载完成消失

20.v-for中为什么要有key

1. key可以提高虚拟DOM的更新效率，
2. 在vue中默认“就地复用'策略，在DOM操作的时候，如果没有key会造成选项错乱。
3. 注意：**key只能是字符串或number**